MODULAR MICROPROCESSOR TEST SYSTEM

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Award of the Degree of

MASTER OF TECHNOLOGY

by

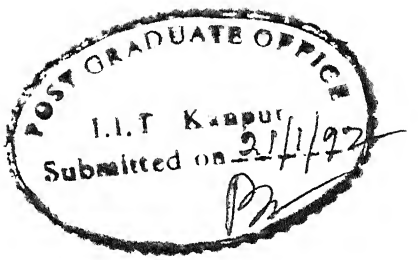
R SRINIVASAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

January 1992

CERTIFICATE



This is to certify that this work entitled

MODULAR MICROPROCESSOR TEST SYSTEM

by R Srinivasan has been carried

out under my supervision and has not been submitted

elsewhere for the award of a degree.

R Biswas

Dr. R N BISWAS

Professor

Department of Electrical Engineering

Indian Institute of Technology Kanpur

ACKNOWLEDGEMENTS

I acknowledge with a deep sense of gratitude and appreciation the help and guidance of my thesis supervisor, Dr. R.N.Biswas, in bringing this thesis work to a fruitful completion. I particularly appreciate the way he managed to obtain for me, at the right instant, the resources to implement this design without hassles.

Since my work had a considerable amount of practical content, it would not have progressed without the cooperation of the technical officers, supervisors and the technicians in the Electrical Engineering department of I.I.T. Kanpur. I thank all the research engineers, too, who had extended all possible help during my thesis work.

I am indebted to Hindustan Aeronautics Ltd. for having given me an opportunity to do M.Tech. at I.I.T. Kanpur, especially through the efforts of Mr. A. Annamalai, Senior Manager (Designs) in AvDB, H.A.L. Hyderabad.

On a personal note, the companionship given to me during the thesis work by Ir. B.P.Shashidhara and Sq. Ldr. Narayan, has been heartwarming. I also express my ratitude to Dr. and Mrs. Sule who helped me out during my personal crisis and rought me to the right track. And it is impossible to forget the contribution of y family in tolerating my eccentrities and separation from them while doing my .Tech.

This thesis report would not have come into such a fine shape without the ntribution of Mr. K.S.Venkatesh. He made up for all my artistic deficiencies ring the preparation of this manuscript. He is a good friend, and I will miss his mpany when I leave I.I.T.

R.SRINIVASAN

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the aim of designing a suitable test equipment for studying any microprocessor in a simulated environment from a commonly available computer, a Modular Microprocessor Test System has been designed with a PC-AT as the host. It has the capability of handling 32-bit microprocessors and has been designed keeping the requirements of CISC processors in mind. From the PC-AT, the user can control signals as well as programme execution on the target processor and study its response through this system. The Test System consists of three modules — PC-Host Interface, General Interface and Target-Specific Interface. The PC-Host Interface module buffers the PC-AT bus from the target interface and is plugged into an expansion slot on the PC-AT system unit. The General Interface contains all the common blocks required to connect the target processor to the PC-AT, while Target-Specific Interface contains the processor-related circuitry. Target-Specific Interface is linked to the General Interface through a common target bus, and up to 16 Target-Specific Interfaces catering to as many as 16 different processors can be connected to it. The Test System has been tested with an 8085A development system and has satisfied the requirements for which it has been designed. This test system is expected to fill a gap in the facilities required for the design of microprocessor-based systems, namely, that of accurate simulation of a variety of processors and their environment for study and pre-design evaluation.

Chapter 1

INTRODUCTION

Among the fastest growing fields in science and technology are the areas of microelectronics and computers. Their conjunction has produced the microprocessor, and most of the innovations in computer hardware can be attributed to the availability of fast and highly integrated microprocessors. The demands of the computer market along with the rapid changes in VLSI design have resulted in processor manufacturers competing with each other in the release of new microprocessors annually, each incorporating more functions than earlier processors. While it is certainly true that the advent of newer processors has enabled the designers of microprocessor-based systems to bring out improved designs to handle a variety of sophisticated needs, the full potential of new processors seldom gets fully exploited. In order to understand the cause of the gap between the capabilities of processors and their utilization in design, let us take a look into the considerations that go in a design process.

In any design, the engineer strives to reach the best balance between performance, cost, design time, reliability and board space. A microprocessor-based design would require him to study the behaviour and potentialities of different processors, calculate the trade-offs between the processors and memory structures

over a wide range of parameters, before selection of the processor and its associated peripherals. This leads to the requirement for a system which allows the designer to accurately simulate an environment for a target processor and to change over to another processor easily. At this stage of design, the designer is not going to formulate software; he is in search of a suitable platform which satisfies the requirements of his system specifications.

Now, we can take a look at the information gap between the processor manufacturer and the applications engineer. The information released by the manufacturer of a microprocessor is accurate, but will not cover all situations in which his product is put to use. Moreover, he will definitely not compare it with other processors, except when it is advantageous to his product. So, the application engineer would like to study the processor signals and functions from his own angle, keeping the requirements of the design in mind, and that too, for as many processors as possible. For this purpose, he should have the freedom to simulate the environment of the processor, and he should be able to *finger* the processor without restrictions . The microprocessor test equipment in the market today cater very well both to software development and testing and to debugging of boards during development, but are not adequate for pre-design evaluation. The proposed test system attempts to fulfill this need of the design engineer.

Besides the discussion on test equipment and the requirements of the proposed test system, Chapter 2 presents the system configuration of the test system with a PC-AT as the host, interfaced with the target processor through a host-target interface. Chapter 3 discusses the host end of this interface, which consists of a card resident on the PC-AT bus. The remainder of the host-target interface comprises two distinct parts — one combining all the common blocks independent of the choice of the target

processor, and the other consisting of the remaining blocks which are specific to the target processor. The common blocks, combined in the General Interface, are described in Chapter 4, with emphasis on the common interface bus to the target processor and its specific interface. Chapter 5 gives the general requirements of the Target-Specific Interface, which are to be satisfied so that it can be connected through the common interface bus to the General Interface; a general block diagram of the Target-Specific Interface is then developed and an example is given for the 68020. Finally in Chapter 6, the method of testing of the test system, using an 8085A development system connected to the common bus, is presented, along with examples of the capabilities of the test system in this environment.

Chapter 2

TEST SYSTEM OVERVIEW

Given the need for a modular test a system for microprocessors, the natural question that arises is whether an exclusive design to meet these needs is required in spite of the available microprocessor test equipments. In the world of test test equipments for microprocessor-based systems, there are emulators, trouble-shooters, logic analyzers and evaluation boards available for the design engineer. It may appear that these systems maybe used directly as educational tools for the desired microprocessors. However, each has its shortcomings which have to be overcome, and have features which can be incorporated in our design. In the following sections we will go into detail about the commercial systems available, and their features from the viewpoint of an educational tool.

2.1 *Evaluation Boards*

Traditionally, microprocessor manufacturers have released evaluation boards for their processors in order to assist the customer in learning its operation and testing his software. Some of them are standalone systems like the SDK-85 and SDK-51 boards of Intel. Nowadays, the trend is to make evaluation boards which can be hooked

up to a standard host system like the PC, or to be operated with a dumb terminal. Evaluation boards do not permit much flexibility in experimenting with new hardware configurations. They control the complete memory and I/O space with their hardware and the user is restricted to running his programme in the designated user area. But these restrictions are justified by the fact that they do not hinder the purpose for which evaluation boards are used, viz, to give the customer a hands-on experience of the new processor.

Evaluation boards offer the user facilities of assembler/disassembler and debug facilities like execution from a fixed starting address, single stepping instructions, multiple breakpoints and examining or modifying registers and memory. They give the user limited control over control inputs to the microprocessor, such as interrupts, rest, halt, DMA control, etc. Also, the status of all control outputs cannot be obtained, since evaluation boards capture trace data for each bus cycle and not for each clock cycle. These drawbacks limit the usage of evaluation boards for a study of the concerned processor, and merely make it a platform for running and debugging software routines written in the assembly language of that processor.

2.2 *Logic Analyzers*

Logic Analyzers are non-invasive tools employed for troubleshooting of logic circuits. They can capture data available on the probes continuously at edges of an internal asynchronous clock, or for a combination of clock inputs to the logic analyzer. The latter mode of operation is used to capture trace data from a microprocessor, using the address and data strobes as clock inputs The captured data is displayed either as waveforms or in bits. The trace data from the processor is disassembled and listed.

The logic analyzer has the advantage of having the capability of capturing events occurring on all control outputs of the microprocessor, including the outputs not synchronised with the processor bus cycle. This feature is a necessity in the study of the microprocessor, since it can give a picture of the relation between assertion of asynchronous inputs to the processor and its output state.

However, the logic analyzer cannot be used as a standalone system in an educational tool, since it cannot control or activate the inputs to the processor.

2.3 *Emulators*

An important tool in debugging of microprocessor-based boards is the emulation of the target processor. Emulation involves combining observation with coordinated control, which is the aim of the proposed test system also.

Two types of emulators are available in the market. They are the In-Circuit Emulator (ICE) and the ROM-based Emulator.

The In-Circuit Emulator substitutes the target processor by a custom-made bond-out device lodged in a pod. The bond-out device give the designer a window through which he can observe and control the processor operation. Since the bond-out device is custom-made, it is the microprocessor manufacturer who comes out first with an emulator for a new processor. Instrument manufacturers come out much later with their emulator for that processor. ICE is available for most standard microprocessors.

The ROM-based Emulator is used in situations like bit-slice boards in which the user replaces the external microcoded control memory with the emulator memory. Through

this memory, the user can control the execution of instructions in the bit-slice processor. However, due to its limited applications, it is not popular.

Current emulators from test equipment manufacturers handle an assortment of processors from more than one manufacturer. However, for each processor, there is a custom-made bond-out device which will have to be installed. This is done by changing the target pod, maintaining the same host system. Such emulators, eg. the Sigma ICE, exercises the target processor at full speed and record its activity without intruding into the target system software or hardware. They differentiate between bus and processor activity, but do not give much information on the current state of the asynchronous or non-deterministic events. Neither do they provide the user with a facility to programme asynchronous inputs from the emulator. This insight can be obtained only by using external circuitry to generate the asynchronous inputs, and capturing the control outputs on a logic analyzer. The difficulty in such a procedure is that the interaction between these inputs and instruction execution cannot be studied. In the case of Intel's I^2ICE, an additional card and software called the iLTA package tries to circumvent this problem. It solves the latter half of the problem by capturing the other logic information from the processor and linking it with the bus activity. However, it still does not permit the user to programme asynchronous inputs to the processor.

Since the target emulator substitutes for the processor, transparency is essential. The emulator should exhibit no differences, functionally or electrically, from the original processor in the board. It should make no demands on any of the target resources such as interrupts and memory allocation. Therefor it should allow the user complete accessibility of all control inputs and memory, either from the board itself, or from the emulator. This requirement, too, has to be met by the proposed test system.

The emulator should duplicate as closely as possible the electrical characteristics of the target processor. It is here that compromises are made by the emulator manufacturers. The bus timings are preserved completely, but loading of bus lines differs from the existing processor. Also, all characteristics of the processor are not duplicated, since the original processor may demonstrate undocumented nonlinearities or performance. This problem is understandable, since the bond-out device is a hardware model for the processor, and will suffer from all the infirmities of such a model. Moreover, since the pod is placed close to the target system, with a flat cable to the main system, loading characteristics on the bond-out device will differ from those of the original processor.

Current generation emulators have become modular. An independent host system such as a PC can be used to control the emulator and display the data captured. They allow change-over from one processor to another by changing the plug-in CPU module.

Emulation offers all the features of evaluation boards. In addition, they permit execution of programmes mapped either on the target board or on the emulator memory. Similarly, data memory and I/O ports can be selectively mapped onto the emulator or onto the target board.

Most of these features of emulators are applicable to our test system design. The critical drawbacks are:

1. Emulators do not permit the user to have control of the asynchronous inputs and do not correlate such outputs with them, or with instruction execution.
2. They substitute the original processor by a bond-out device which differs in electrical characteristics and may differ in functional transparency.
3. These custom-made bond-out devices are much costlier than the original

processors themselves and thus make the entire system expensive. The time overhead maybe 2 years from the launch of a new processor, for its emulator pod to be released for a universal type emulator, by the test equipment manufacturer.

2.4 *Simulation as an Alternative*

An alternate solution in the study of microprocessors is simulation. A cost-effective analysis of a complex system can be done through computer simulation. Simulation consists of deriving an abstract model from concrete situations, and interpreting the model to provide insight into the original situation. For simulating microprocessors, the inputs for constructing a model are available from the parameters given by the processor manufacturer. Most of the events occur at fixed time intervals, especially those concerned with the bus cycle. Such events are deterministic and can be simulated by a time-slice model. However, events such as interrupts, bus arbitration and DMA control are non-deterministic. Such events require a discrete-event model.

Software simulation of microprocessors has not found a wide application in design of microprocessor-based systems. There is indeed a requirement of simulation of various processors during pre-design evaluation, when the choice of a processor has to be made. Software simulation has an important drawback which prevents its use here. The model is simpler than the phenomenon it represents and its representation is approximate. Therefor, it is accurate only to a limited extent. Evaluation for the purpose of a reliable comparison of different microprocessors requires accurate data to be obtained in various test patterns, about the performance of the processors involved. Also, software simulation is not able to provide the hardware features, eg. speed, loading, etc.

As a result, hardware modelling is used in pre-design evaluation of microprocessors for specific applications. Hardware modelling combines hardware and software into one modelling solution. The simplest way is to use the actual device to model its own behaviour. This gives the fastest and most accurate model for pre-design evaluation. Such modellers can format inputs from the simulator, evaluate device behaviour and return the resulting device outputs with their timing information. Unlike software models, hardware models are functionally transparent and can model the device characteristics as it occurs, including non-linearities and undocumented characteristics. Such a model can be built and operated in a short time without the cost overhead of emulators.

2.5 *The Structure of the Proposed Test System*

From the study in the previous sections of various methods for testing microprocessor systems, we can get a clear picture of the requirements of our test system.

The system should have the signal-grabbing capability of the logic analyzer, acquiring the signals with reference to the bus cycles as well as to the clock cycles. It should have control over programme execution, as in an emulator, with its debugging capabilities. It should possess the ability of the hardware simulator to provide the true electrical environment of the microprocessor, which is fully accessible to the user. It should also have the feature of universal emulators to provide a general platform from which different microprocessors can be exercised, by merely changing the target pod. But the objective of the test system is not software development, as is done by emulators. Neither does it have to debug microprocessor-based boards. Instead, the aim is to simulate the microprocessor along with its hardware environment, for

study by the user. It need not operate at the maximum speed of the microprocessor, as can be handled by emulators.

The proposed test system can combine all these features by using the actual microprocessor to execute the code, generated off line by the host system from the user programme, and by providing a simulated environment for the target processor. Let us next examine how such a test system can best utilize a standard host system for studying a target microprocessor. To make it modular, the host-target interface has to be split into two parts:

1. A General Interface, common to all microprocessors.
2. A Target-Specific Interface, between the processor and the General Interface.

The requirements of these two parts of the the interface are taken up next.

2.6 *The General Interface*

As a first step, the General Interface should have a trace buffer, which can be written into by the target processor at full speed, based on its bus cycle and clock edge. This trace buffer can be emptied by the host at a slower speed. The next requirement is that it should have the registers to write data, address and controls for the processor, with the processor in control of enabling the outputs of these registers. There should also be a RAM which can be loaded with the test code by the host system. Subsequently, the processor should be able to execute this code at full speed. Since the aim of the test system is not software development and it does not have to run the user programme completely at full speed, the RAM area can be kept small. But the same RAM will have to provide access to the complete memory space of the target

microprocessor. To meet this requirement, the concept of a *tag address* and *page* will have to be introduced.

The user programme is divided into pages, each page being the size of the RAM. The upper bits of the address corresponding to that page, are fixed and constitute the *tag address*. Each time the target processor accesses the RAM, the tag address which is preloaded by the host system, is compared with the upper bits of the address. If they are not equal, the current bus cycle is suspended till the new page and tag address are loaded by the host system.

Since the tag address register and a hardware comparator are general requirements, they form parts of the General Interface. The other registers that are obviously general in nature are the Command and Status registers. The commands to run the target processor and to rest the trace buffer from the host will have to be written through this Command register. Similarly, the status of the target processor operation will have to be routed through the Status register.

2.7 *The Target-Specific Interface (TSI)*

The Target-Specific Interface will have to be connected through a standardised *target bus* to the General Interface. Circuits which are common between TSIs will have to be transferred to the General Interface, retaining only the target specific ones. Obviously, generation of signals like Clock, Ready, Transfer Acknowledge etc. is part of the TSI. But to achieve an efficient separation between the two interface circuits, a comparison of different microprocessors for their common features will have to be made.

2.8 *Allocation of Signals between the General and Target-Specific Interfaces*

Most leading CISC microprocessors are from two IC manufacturers - Intel and Motorola. A study of the signals from different processors, grouped according their functions, can help in bringing out the similarities and differences between them. Four upper-end 32-bit CISC processors have been selected for this study. They are the 80386 and i486 from Intel, and, the 68030 and 68040 from Motorola. The 80386 and 68030 are the last of their generation of microprocessors. Earlier 8-, 16- and 32-bit microprocessors have hardware configurations which are subsets of these processors. The i486 and 68040 are even more complex than their predecessors, with new techniques of implementing existing functions, to obtain higher speeds. Hence, these four processors were selected to obtain a worst-case estimate for the allocation of signals between the General and Target-Specific Interfaces. This study does not cover RISC, bit-slice processors and microcontrollers. This is true of the design of the test system too. Details of the comparative study are given in Appendix A The conclusions that can be drawn from this study are given below.

Each company, i.e. Intel and Motorola, has its own interfacing method for each function, making them mutually incompatible. There is much in common between processors of the same company, but the i486 and 68040 break away from the past trends. There are differences in interface even for similar signals. For example, Intel processors give out the decoded byte enable signals to select bytes of data. The Motorola processors expect the designer to decode the byte enable signals from the size information SIZ1 and SIZ0, and the address bits A1 and A0.

The differences extend to the facilities offered in each function. For example, an external HALT signal can force the Motorola processor to halt execution. Intel processors allow only a software halt, through the HLT instruction. Above all, the types of functions vary between individual processors. The latest processors, i486 and 68040, provide the additional function of cache-line invalidation by snooping on the bus when an external bus master is in control. A built-in facility is also provided, with the 68040 implementing an IEEE P1149.1 Test Access Port.

From these considerations, it is obvious that a general interface can provide only a set of programmable control inputs to the target. The TSI will have to contain circuitry to condition these outputs, as per the microprocessor being targeted. Similarly, the control outputs from the target processor will have to be conditioned in the TSI before they are acquired in the trace buffer.

The data bus, too, is not free from its requirements for individual target interfacing. This is a consequence of different methods of implementing dynamic bus sizing in the microprocessors. In the case of i486, data should be on the addressed data pins, irrespective of the bus size. In the others, data should be on the least significant parts of the bus for byte and word transfers. This leads to a requirement for external swap logic in the TSI to permit proper access to the general bus.

The capture of target information and suspension of the target processor operation will obviously have to be a part of the TSI. The generation of breakpoints, which are dependent or concerned with the processor, have to be a part of the TSI. The causes of break can also vary and should be read from the TSI through the Status register in the General Interface. Even the generation of the common breakpoints, i.e., address out of range and single-step execution over, have to be a part of the TSI,

since they interact directly with the processor signals. The only common breakpoint that can be situated in the General Interface is the trace buffer full condition, as it is independent of the processor under test.

The other circuits which are processor-specific, are the read and write controls to the trace buffer, RAM and input registers to the target processor. In addition, there should be provision for identification of the target board and activating the match flag to the host. This will let the processor know if the required target processor board is connected or not. Also, such a flag can be used to disable outputs of the target board, when inactive, permitting multiple target boards to be located on a general target bus and studied or simulated one at a time, through the controls from the host.

Another information yielded by the comparative study is the size of the general target bus between the Target-Specific and General interfaces. Control signals can be classified into three types - inputs, outputs and bidirectional signals. Again, the control outputs can be sub-divided into signals synchronised with the bus cycle, and signals to be captured on a clock edge. From the comparison between the four microprocessors, a maximum of 18 control inputs, 13 outputs to be latched on the bus cycle, 9 on the clock edge and 7 bidirectional signals has to be allocated on the bus. Therefore, we can allot 24 lines for control inputs, 15 outputs to be latched on the bus cycle, 15 outputs to be latched on the clock edge and 8 bidirectional control signals. Besides these, 30 lines are required for address and 32 lines for data, both being bidirectional. Therefore, the trace buffer width is 100 bits. The width of the target bus is has to be 124 lines at least. Additional lines have to be allotted for the read and write controls, status and breakpoint request from the TSI to the General Interface. Commands from the host through the General Interface will occupy a few

lines on the target bus. Using the size of the bus from the the TSI as one of the parameters, we can now proceed to the selection of a suitable host.

2.9 *Choice of a Host*

The host system has to run independent of the target microprocessor, while presenting a common target interface bus of 124 lines with additional lines for control of the boards and for status of the TSI. It will have to be interfaced with a monitor and keyboard for the human interface. Also, it should provide access to assemblers/disassemblers, cross assemblers and compilers. Three options are examined below.

1. Microcontrollers provide a large number of port pins, of the order of 24 pins in microcontrollers like 8748 and 8751, and include peripherals like timer, event counter and ADC. Since the host system has to interface with over 100 lines of the target bus, the microcontroller has to be operated in the expanded mode with external address and data buses. In addition, interface with keyboard, monitor, floppy and/or hard drives has to be provided, along with their software.
2. The Personal Computer (PC) is a low-cost computer available in the market today with a versatile set of application packages available for the user. The basic system configuration consists of a monitor, a keyboard, and a floppy drive with the main computer. It has a serial port for data transfer and a parallel port for the printer interface. Its hardware is well- documented and it permits addition of prototype cards on its bus. The basic PC is slower than most microcontrollers and definitely much slower than a workstation.
3. The workstation is a fast standalone computer with capabilities of handling multiple terminals concurrently. It offers all the facilities of the PC. Its speed

makes it an ideal tool in number-crunching operations like simulation, image processing, etc. However, it is much more costly than the PC.

A comparison of the three options rules out the microcontroller immediately. The microcontroller is not a standalone system like the other two, and the rest of the required system will have to be designed around it. Also, it offers no advantages in handling the wide target bus, over the other two. In all the cases data will have to be exchanged between the host and the target in a multiplexed manner.

The workstation can be employed as a central host linked to different target ports separately through parallel ports. Different target processors can be studied from different terminals hooked to the common host. But it would not be possible to study the same target processor module from two different terminals, since this will interfere with the hardware controls and the trace data. So a separate target module has to be allocated to every user. This leads to the requirement of separate host and target boards for each user. The PC fits the bill nicely, being much cheaper than a workstation. It is slower than the workstation, but this will not matter for the off-line operations carried out by the host on the target interface. Though the workstation is fast, it will still not be able to operate the target processor at full speed.

The PC, PC-XT and PC-AT have a serial port for communication and a parallel port for operating the printer. If we have a trace buffer with a depth of 1K words, each of width 100 bits, we will have to transfer about 13 Kbytes of data from the trace buffer to the host, each time it is full. So use of the parallel and serial ports is ruled out. The only option left is to use the PC bus itself for communication with the target. Out of the three members of the PC family, the PC-AT is preferred for this requirement, since it has a 16-bit data bus and a faster system microprocessor.

Since the target bus from the general interface has a large number of lines, the General Interface board cannot be made as a plug-in card for the PC-AT. It has to be situated outside the host and linked to the PC-AT through flat cable. Buffers and some additional circuits will have to be situated on the plug-in card in the PC-AT, the other side of the buffers being linked through the cable to the General Interface cards. The resulting system configuration is shown in Figure 2.1. The PC-Host Interface, General Interface and Target-Specific Interface will be discussed in detail in subsequent chapters.

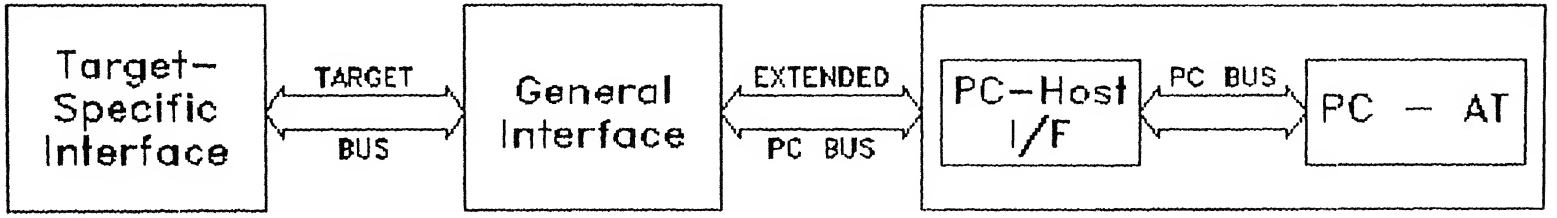


Figure 2.1: Block Diagram of Proposed Test System

Chapter 3

PC-AT TO HOST CARD INTERFACE

PCs permit the user to design prototype cards and plug them into the PC bus. It reserves addresses in the I/O space and interrupt request lines for the prototype card. In addition, PC-ATs allow memory and I/O devices access to the 16-bit data bus. These features have been used in the design of the interface card. The following sections will examine the design of this card in detail, with reference to the block diagram in Figure 3.1. The detailed schematic diagram is available in Appendix D. A brief description of the PC-bus signals used in the design and the Extended PC-bus signals is given in Appendix B.

3.1 *Buffers*

Buffers are used on all the lines required to extend the PC bus to the General Interfaces. The buffers on the data lines SD0 to SD15 are bidirectional with the buffered IOR (BIOR) controlling the direction of data flow. The data buffers are enabled whenever the prototype I/O area is selected. The other buffers are unidirectional, with the control and address buffers permanently enabled.

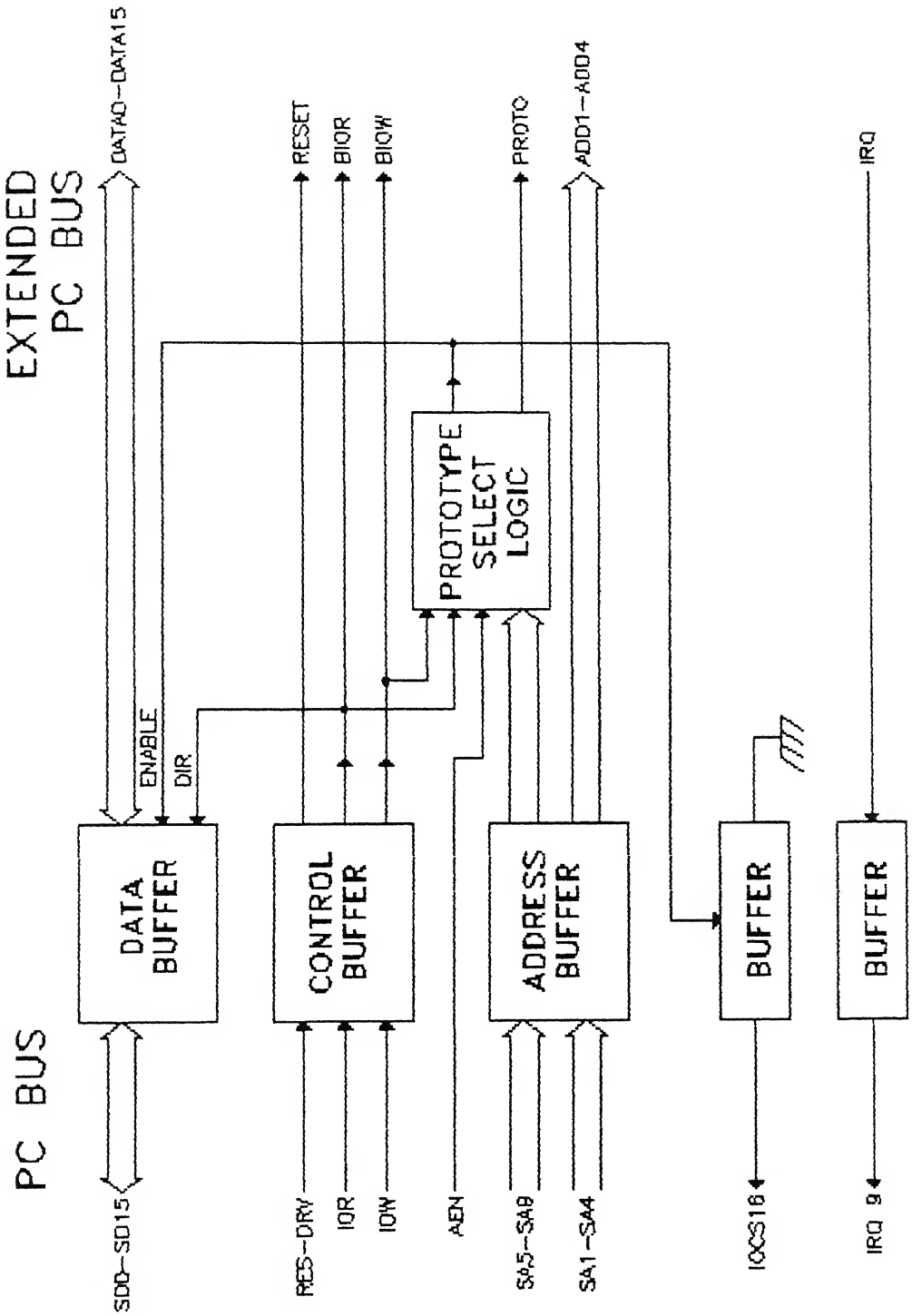


Figure 3.1: Block Diagram of PC-AT — Host Interface

3.2 *Prototype Selection Logic*

The I/O address map of the PC-AT allocates addresses in the range 300 hex to 31F hex to the prototype card. The lower five address bits SA0 to SA4 have to be decode to select the individual peripherals, with the rest of the address bits generating the gating signal for the decoders. IOR and IOW signals are used as enable signals for the decoders also.

Here, the main aim is to simplify the interface and reduce the number of signals between the two modules, viz Interface card and General Interface, on the *Extended PC Bus*. So the address decoder is placed locally on the General Interface with buffered address bits SA1 to SA4, buffered IOR and IOW and the gating signal PROTO. Since the I/O peripherals on the General Interface can be addressed as 16-bit devices located at even addresses, the address bit SA0 is not required by the address decoder.

Since the I/O address map of the PC-AT is from 0 to 3FF hex, we have to decode the address bits SA5 to SA9 only, to generate *PROTO*. This is the reason for the selection of the I/O space instead of the memory space for the host peripherals, since the latter will require decoding of all address bits upto SA19. The generation of PROTO has AEN from the PC bus as an input also, allowing the General Interface to respond only to the system microprocessor, and not to a DMA access. PROTO is active low, with the Boolean expression:

$$\overline{PROTO} = SA9 \cdot SA8 \cdot \overline{SA7} \cdot \overline{SA6} \cdot \overline{SA5} \cdot \overline{AEN}$$

The prototype selection logic also generates the enable signal for the data buffers and the IOCS16 buffer. PROTO is gated with the BIOR and BIOW to assert the enable signal when a read or write to the prototype area in the I/O space takes place.

IOCS16 is an active low signal indicating to the system microprocessor that the I/O device selected for the current data transfer is a 16-bit device, requiring only one wait state. This is the default number of wait states inserted by the system hardware in each bus cycle. This gives a 240 ns read or write pulse, which is sufficient for access to TTL registers and buffers. The need for the IOCS16 buffer arises because IOCS16 has to be driven LOW, whenever there is a read or write to General Interface, and floated otherwise to meet the requirements of the PC bus.

3.3 *IRQ Buffer*

The PC-AT allots the IRQ9 for users with the software redirected to Interrupt 0A hex (IRQ2) through a software interrupt instruction. IRQ2 in the PC-AT handles the interrupt from the slave interrupt controller to which IRQ9 goes, and hence is free for user software. Thus IRQ9 is compatible with IRQ2 available in the PC and PC-XT. Other reserved interrupt request lines are IRQ10, IRQ11, IRQ12 and IRQ15. Of these, IRQ9 has the highest priority. For these two reasons, IRQ9 has been used in this design.

The generation of the interrupt request is done in the General Interface with a single *IRQ* line coming to this card. Here it is inverted in the buffer to give an active HIGH IRQ9 for the PC bus.

This card has been implemented as a PCB which can be plugged into any PC-AT system-unit expansion slot except number 1 or 7. It has a 25-pin D connector plug and a separate Ground terminal for taking the Extended PC bus out of the PC-AT.

Chapter 4

GENERAL INTERFACE

The General Interface consists of the interface circuitry between the PC-AT and the target microprocessor that are common to all target processors. On one side, it has communication with the PC-AT over the extended PC bus from the PC-AT Host Interface card, and on the other side it has communication with the Target-Specific Interface (TSI) through the target bus. The following sections give a detailed description of the blocks comprising the General Interface with reference to the block diagram given in Figure 4.1. The signals on the target bus are listed in Appendix B.

4.1 *Bus cycle Trace Buffer (BTB)*

This trace buffer stores the outputs of the target microprocessor that are synchronised with its bus cycle. It requires a write signal, STB0, from TSI. BTB has to store the address bits A2-A31, data bits D0-D31, control-bits COP0-COP14 and the bidirectional control bits CI00-CI07.

The operation of writing into the buffer has to be compatible with the speed of the target processor. This necessitates a fast FIFO to store the trace information. The

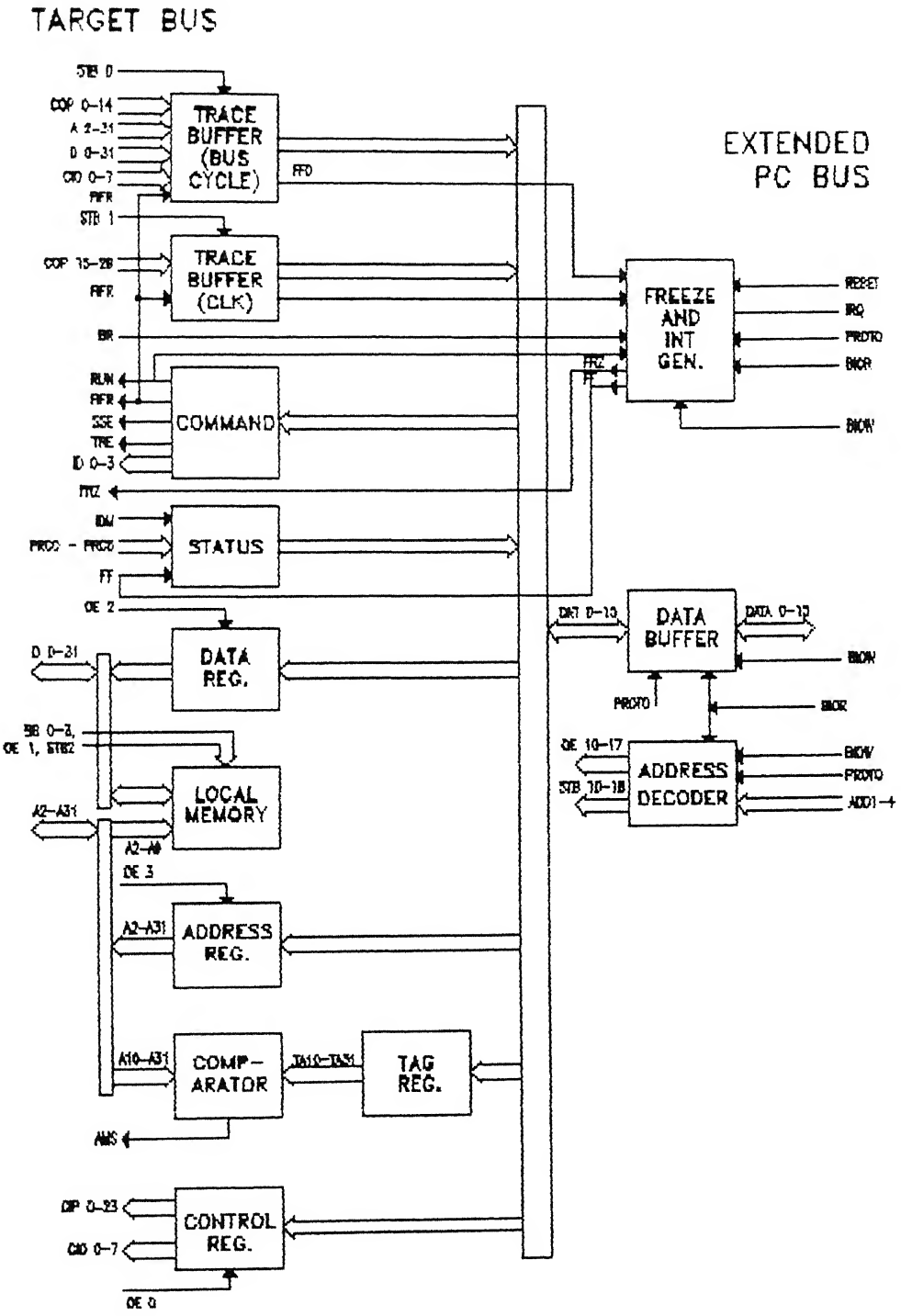


Figure 4.1: Block Diagram of Host Cards

depth of the FIFO is not a critical parameter, since our proposed test system is not being designed to cater for the running of a large user programme at full speed. Commercial FIFOs of size 1024 x 9 and a speed of 35 ns are commonly available and have therefore been chosen for this design. The width has to be expanded by connecting FIFOs in parallel with a common write control STB0 and a common reset control FIFR.

Since the memory is being operated in a paged mode, any page change will suspend execution of the programme on the target processor. So the upper bits of address are constant during execution and need not be stored in a FIFO. Only the final value of the address is required, to determine the location of a new page. Considering a local memory of size 256 x 32, only the address bits A2-A9 have to be acquired in a FIFO. The rest of them can be written into a register, using the STB0 signal. So BTB has 8 FIFOs and three 8-bit registers for capturing the trace information.

All these FIFOs and registers are read by the PC-AT through its data bus and are controlled by individual read signals. BTB gives two status outputs to the host system; one is the Full Flag, FF0, and the other is the Empty Flag EF0. FF0 is used to freeze the running of the target processor and to generate an interrupt request to the PC-AT, so that it can be read and emptied. EF0 is read through a buffer along with the FIFO contents. It allows the PC to stop reading the FIFOs once they are empty.

4.2 *Clock cycle Trace Buffer (CTB)*

This trace buffer is similar to BTB, except for the signals connected to it. STB1 is the write signal from TSI and is synchronised to the processor clock. The inputs to CTB

from the target bus are COP15-COP29, which are not synchronised to the bus cycle. Therefore, this buffer has only two FIFOs. Like BTB, CTB also has two status signals corresponding to the Full Flag and Empty Flag conditions; these are denoted by FF1 and EF1. Like FF0, FF1 also freezes the running of the target processor and generates an interrupt request to the host system; EF1, read through a buffer, indicates that CTB is empty.

As CTB captures information on every clock cycle, while BTB captures it on every bus cycle, CTB will be full first. So FF1 will be the cause of an interrupt request rather than FF0, in the absence of other break-points.

4.3 *Command and Status Registers*

The *Command Register* is the port through which the host system controls the running of the target processor, selecting the break-points to be enabled and the target processor to be exercised. The different controls indicated in Figure 4.1 are described below.

The *RUN* signal allows the processor to start running when a positive edge is given, and to continue running as long as this signal is HIGH or a break-point is reached.

The *FIFR* signal resets all FIFOs by giving a low pulse. During normal operations, the FIFR is held HIGH. To disable the storing of data in the trace buffers, it can be kept low.

Single-Step Enable or *SSE* is given to TSI, so that detection of this break-point is asserted after the execution of the current bus cycle, if it is held HIGH. Otherwise the single-step execution break-point is disabled.

The Trace Enable signal *TRE*, enables all break-points in TSI, if HIGH. When this signal

is low, break-points from TSI are to be disabled, including single step, irrespective of SSE

By making TRE and FIFR LOW and asserting the RUN signal, the test system can run the target processor without break-points, allowing the user to study waveforms on an oscilloscope. However, the trace information will be lost.

The final sets of control from the Command register is the Identification code, *ID0-ID3*. It allows upto 16 different target boards to be plugged into the target bus.

The *Status Register* allows the host system to read the status of the target system. It has two sections, break-point cause and ID match status. The break-point cause contains two subfields: FF from the General Interface and the Programmable Cause *PRC0-PRC5* from the TSI. Though FF0 will not be asserted usually, it is still used in generation of FF to take care of unusual conditions. FF is asserted if either FF0 or FF1 is asserted. If the number of break-points exceeds 6, PRC0 to PRC5 can be encoded. The maximum number of target break-points that can be identified is 63, since the condition of all 1's is reserved to indicate the absence of break-points.

The other signal that the host system has to read from Status Register is IDM, which signals that the identification code ID0-ID3 has matched that of TSI. It is an active LOW signal.

4.4 *Data, Address and Control Input Registers*

When data has to be written on the target data bus for a single-step operation in the case of data memory or I/O access, it should be possible to write just one value from the host system instead of transferring the complete page to the local memory. For

this purpose, a separate *Data Register* has been incorporated, permitting the transfer of a single 32-bit data word from the host system to the target bus. Read control of this register is exercised from TSI, by the signal OE2.

The *Address Register* allows a single address value loaded by the host to be driven onto the target bus, enabled by OE3. It can be used for testing DMA and bus snooping functions, if provided by the target processor. In addition to these functions, the Data and Address Registers are used by the host system to write into the local memory, as discussed in the next section.

The control inputs for the target processor CIP0-CIP23 are written by the PC-AT into three 8-bit registers. The outputs for the target bus are permanently enabled, since they are not bidirectional. The lines CI00-CI07 are outputs from a single register, written into by the host. Since the outputs go to bidirectional lines on the target bus, the signal OE0 from TSI is used to enable it.

4.5 *Local Memory*

A Read/Write memory or RAM is required in the test system so that the test programme can be written off-line from the host system and subsequently executed by the target processor at full speed. The available memory space in a 32-bit microprocessor is 4 gigabytes which have to be fully accessible from the host system. However, since the objective of this test system is not software development, we can afford to have a small local memory with the 4-gigabyte memory space acting as a virtual memory. The size of the local memory space chosen from these considerations is 256 x 32, or 1 kbyte.

The RAM has to be fast enough to satisfy the needs of the target processor. An access time of 35 ns is sufficient for its requirements since it need not be run at the maximum speed. The target processor has access to this local memory in the run mode, through its address lines A2-A9 and data lines D0-D31. BS0-BS3 from TSI select individual bytes in the data word. The read and write controls from the target processor are OE1 and STB2.

A new page has to be loaded into the RAM whenever the current bus cycle requires a different page from the existing one in the memory. This is signalled by the page-change break-point from TSI. The break-point for page change suspends target operation and generates an interrupt to the host system. The address of the new page is available in the trace buffer.

The next step is for the PC-AT to write into the RAM. Valid address and data have to be present on the inputs to the RAM before giving the write signal from the PC. Since it can write onto the address and data lines only through the respective registers discussed in section 4.4, they are used as intermediate registers for the RAM. First the A2-A9 bits of the required address are written into the address register. The data will have to be written in two steps to the data register, first D0-D15 and then D16-D31. The final strobe pulse to the data register is used to write into the RAM, which takes in the data present in the data registers at the positive edge. The dual control of the common enable signals is discussed in the next section.

4.6 *Dual Control of Enable Signals*

When the PC-AT has to write into the local memory, the outputs of the Data and Address Registers have to be enabled and OE1 disabled. So the signals OE1, OE2, OE3, SB0-SB3

and STB2 are taken over by the General Interface through a buffer which is enabled when the target processor operations are suspended. A similar buffer on TSI disables them during this period. OE1 is negated, OE2, OE3 and SB0-SB3 are asserted, while STB2 is pulsed by the final write pulse from the PC-AT, used to write into the Data Register.

In this way, we have tried to make effective use of available circuitry in the General Interface to implement a dual-ported RAM, accessible from the PC-AT and the target processor, but with non-overlapping controls.

4.7 *Tag Comparator*

The tag comparator is made up of a hardware *Comparator* and a *Tag Register*. This block is used to generate the *AMS* signal, indicating whether the address on the target bus matches the tag value in the register or not. The AMS signal is sent to TSI, which has to gate it with the processor address or data strobe to validate this signal before using it.

The Comparator compares the address bits A10-A31 on the target bus with the corresponding outputs from the Tag Register. The tag value of the current page is the upper 22 bits of the address, and is written by the host system when it loads the page into the local memory. The outputs of the Tag Register are enabled permanently. This circuit plays a key role in the generation of the page-change break-point in TSI.

4.8 *Freeze and Interrupt Generation*

The host system initiates the running of the target processor by asserting RUN in the Command Register. The RUN signal cannot be used directly to control the target

processor, since it is an output of the host system port which reacts too slowly to stop the processor activity at a precise point. So a separate freeze signal, FRZ, is sent to TSI for this purpose. When RUN is asserted, FRZ is negated and the break-point logic signals the stopping of the processor activity by asserting FRZ. The break-point logic in TSI is combined into one signal called *BRK*, which signals the occurrence of break. The combined FIFO full flag, FF, and BRK are combined in the General Interface. When either one of them is asserted, indicating a break condition, the transition is used to assert FRZ. FRZ is negated only after RUN is asserted again. FRZ is asserted when a reset pulse is applied on the PC bus, or if RUN is negated. This enables the PC-AT to have control over the running of the target processor directly through the RUN command, and make it suspend execution when the PC is being reset during a power outage or user intervention.

Interrupt generation for the PC-AT requires that the break-point logic used to generate FRZ assert the IRQ line, while the entry into the ISR should negate it. Entry into the ISR is assured when the PC accesses its ports in the General Interface, after the IRQ signal is asserted. Therefore, IRQ is negated when PROTO is asserted and either BIOR or BIOW is asserted. It prevents re-entry into the ISR for the same request.

The IRQ line is routed through a buffer to the extended PC bus. This buffer is disabled on power-on to the General Interface, to prevent spurious interrupt requests from causing the PC to hang. The buffer is enabled after the first write operation to the Command Register from the PC.

4.9 *Address Decoder and Data Buffer*

These blocks control the interaction between the host system and the General Interface. The Address Decoder controls read/write operations of all FIFOs and registers in the General Interface. This block actually consists of two separate address decoders, one to generate the read controls OE10- OE17, and the other to generate the write controls STB10-STB18. This scheme permits I/O addresses to be overlapped for read and write operations, allowing more ports to operate in the same I/O space. The I/O address map, with the corresponding controls is given in Appendix C.

The PROTO signal from the extended PC bus is used as the gating signal for the Address Decoder. BIOR and BIOW are the enable signals for the two decoders generating the read and write controls respectively. The four address lines from the extended PC bus, ADD1 to ADD4, are the inputs to the Address Decoder.

The data buffers are bidirectional and channelise the common data path between the FIFOs and registers to the extended PC bus. The data buffers are enabled whenever PROTO is asserted, and either BIOR or BIOW is also asserted. The direction of data flow is given by BIOR.

4.10 *Implementation*

The implementation of the General Interface was done with due consideration to the requirements of the target bus. The target bus has around 150 lines and therefore requires a suitable motherboard into which different TSIs can be plugged along with the General Interface. The motherboard used for the VME bus is suitable for this purpose, as it has around 180 distinct connections excluding the supply lines. It could

also accommodate the Extended PC bus connections to the General Interface. Therefore, the size of the card for implementation of the General Interface is as per the VME standards which provides for 9.2"x6.3" boards. In this area it is not possible to accommodate the entire circuitry required to implement the General Interface. So, it has been partitioned into two cards, *Host1* and *Host2*, each conforming to the VME dimensions. The detailed circuit diagrams are given in Appendix D.

It should be noted that the Host1 and Host2 cards conform to the VME standards only to the extent of their physical characteristics. Electrically they are incompatible to the VME bus, and if plugged into such an environment, can cause damage to all cards on it. They have to be used on the target bus only.

In the next chapter, we will examine the Target-Specific Interface with respect to the signals on the target bus and the target microprocessor.

Chapter 5

TARGET-SPECIFIC INTERFACE

Since the Target-Specific Interface (TSI) is unique for each microprocessor, it is possible only to list out its general requirements for interfacing with the host cards. These requirements are covered in the first half of this chapter, while the latter half is devoted to an example of a TSI design. Reference is made to signals on the target bus described in Chapter 4, listed in Appendix B for easy reference.

5.1 *Processor Identification*

Each TSI should have a unique 4-bit Identification Code (ID) preset on switches. The value of this ID has to be compared with the code on the ID lines of the target bus. If they match, the IDM input has to be asserted to the Status Register; otherwise it should be floated. It should give an ID Enable (IDE) signal for internal use in TSI.

5.2 *Inputs and Outputs*

Loading by TSI on any signal line should be restricted to a standard TTL load. This applies both to the inputs to TSI and to the bidirectional signals connected to TSI. All

outputs from TSI should be driven by tri-state drivers capable of sinking 20 mA. This requirement has to be satisfied so that multiple TSIs can interface with the target bus simultaneously.

Drivers for the BRK, IDM and PRC lines are to be enabled only if the IDE signal is asserted. Drivers for the rest of the outputs are to be enabled only if IDM is asserted and FRZ is negated. This condition allows only the selected target processor to have control of the target bus, when permitted to run by the host.

5.3 *Usage of Inputs and Outputs*

The data and address lines of the target bus are used as in normal transactions between a processor and its local memory, when the processor is being run. The current page being run is mapped onto a 256 x 32 RAM, for 32-bit bus access. In the case of 16-bit and 8-bit bus accesses, it is possible either to use the complete space available in the RAM or to use only the section of the RAM corresponding to the current width of the data bus. This flexibility allows the TSI designer to select the local memory structure which will simplify his design and reduce delays. Additional address bits A0 and A1, if available, can be stored in the Bus cycle Trace Buffer (BTB) as a field in the control outputs.

The control outputs should be taken out without conditioning, as far as possible. This will help get a true picture of the target processor activity. A linkage should be provided in TSI, between the trace information stored in every bus cycle and the information for every clock cycle. It will require an indication in the Clock cycle Trace Buffer (CTB) that the current clock cycle has corresponding bus cycle

information also. The STB0 signal itself could be given out as a COP latched in each clock cycle for this purpose.

The control inputs, the CIP lines, have to be latched at the beginning of each run cycle, when FRZ is negated. By latching them, the inputs to the processor are kept stable and allowed to act at a well-defined point, under the control of the host. Again, like the control outputs, it is preferable not to do any extra conditioning of these signals. In some cases where there is a requirement for a short pulse, the transition on the corresponding CIP line has to be sensed, and the pulse generated with the help of the processor outputs and its clock.

The bidirectional control signals, the CIO lines, can be used solely as control inputs or as control outputs when they are not required as bidirectional signals in TSI. As control outputs, they can be assigned only to bus-cycle related signals, since they are stored in BTB. When they are used as control inputs, OE0 will have to be asserted during the run cycle.

It is also possible to employ extra CIP signals in TSI for any use which does not involve the target processor directly, but gives the host extra controls on TSI, like selecting break-points to be enabled. This is true of extra CIO lines also, if they are used as inputs only.

5.4 *Generation of Read and Write Controls for the Target Bus*

The three write lines STB0-STB2, the four read lines OE0-OE3, and the four byte select lines SB0-SB3 have to be generated by TSI when the processor is running. They are

bidirectional, except for STB0, STB1 and OE0, and will have to be floated when IDE is negated or FRZ is asserted.

STB0, the write signal for BTB, has to be asserted when valid address, data, COP and CIO signals are available on the target bus for the current cycle. Therefore, it will be related to the address strobe or data strobe from the target processor, maybe with a delay. The trace information from BTB may come from latched signals from the corresponding outputs of the target processor, so that stable data is ensured during the assertion of STB0. STB1 is the write signal for CTB and will have to be generated from the processor clock. Both these signals will have to be gated by BRK and FRZ. Care has to be taken that the trace information for the current cycle is stored in the trace buffers, during a bus-cycle related break, like page change or single-step execution completion, before BRK is asserted.

OE1, STB2 and the SB signals are the control signals going to the RAM on the host cards and are related to the read, write and byte select signals from the target processor. The byte select signals may be directly available from the processor, or will have to be decoded from A0, A1 and the size information.

OE0 is required to enable outputs from the CIO register in the host card. OE2 is required for read operations, either from data memory or from I/O peripherals, during a single-step execution. OE3 is to be used when the processor bus has been floated, for reading the address from the Address Register. It is used to study the bus snooping function, if provided by the target processor.

5.5 *Break-point Logic*

TSI has to incorporate provisions for generating the break-point indication BRK, along with the PRC signals under various conditions. The minimum break-point conditions required from TSI are for sensing the page change and for detecting the completion of a single-step execution. Break-point circuitry to detect halt, reset, interrupt acknowledge, bus idle state, etc. can be added, and the choice is left to the TSI designer. The outputs of all the break-point detection circuits should be combined to give a common BRK signal to the host. On detection of a break-point, BRK has to be asserted till FRZ is asserted, and negated before the start of the next run cycle. The status of the individual break-points detected should be available as stable outputs till the next run cycle, so that the host system can read the cause of the break through the PRC lines.

There are two extreme manners in which PRC lines can be used to indicate the cause of the break to the host. One option is to provide linear select, permitting one to handle upto 6 break-point conditions. At the other extreme, all 6 PRC lines can be encoded, permitting upto 63 hardware break conditions to be detected. As the latter scheme cannot provide information about multiple breaks, depending on the break conditions TSI is required to report, one can consider an intermediate scheme with a mixture of encoded lines and linear select.

The break-point logic has to perform two operations – generate BRK for host intervention, and generate the required controls for suspending target bus activity – when a break-point is detected. Suspension of target processor activity is discussed in the next section. At the moment, it is sufficient to say that the two operations are distinct, even though they are related. This is so, since the STBO control for storage

of trace information in BTB requires the assertion of BRK to be delayed till the current information is loaded into the trace buffers, whereas the processor may require its activity suspended earlier.

The Address Miss signal (AMS) from the Comparator on the General Interface can be used by TSI to detect the requirement for a new page. The responsibility for validation of AMS lies with TSI, since the comparator merely compares the available signals on A10-A31 of the target bus with the tag address. So AMS will have to be gated with a delayed address strobe or the data strobe in TSI, for detection of page change.

Single-step execution should permit the current bus cycle to be executed by the processor and then assert BRK to suspend the next cycle. If possible, this break-point logic should assert BRK after the current instruction, by sensing the start and end of an instruction. Otherwise, software can achieve the same effect by performing multiple single-step execution on the target processor, with the correct end-points determined by disassembly of the executed codes.

Two controls are available for the host to enable or disable break-points in TSI. The Single-Step Enable, SSE, should be used to enable the corresponding logic. Trace Enable, TRE, should be used to disable the complete break-point logic, irrespective of SSE.

5.6 *Suspension of Target Processor Activity*

In the present context, the suspension of target processor activity is being contemplated only in order to enable the host system to intervene before the target

processor executes its code. Clearly, this rules out options like halt, reset, interrupts and jumps, because these will all cause branching to a fixed address or a program-definable address. On the other hand, forcing the processor to go into a wait state, by using data acknowledge or ready lines, enables the processor activity to be suspended in terms of its bus cycles, and permits the processor to resume operation after the host is ready to continue. One has to note, however, that this technique suffers from the limitation that it will have no effect on functions not related to the bus cycle, like bus arbitration, DMA control, etc. The outputs related to these functions will remain in the previous state until they are forced to change either due to a change in the inputs, or according to a pre-defined processor routine. This implies that the processor is running always, irrespective of host control. The only way to halt all functions is through execution of halt or reset, which is not feasible in this test system. The processor signal, Data Acknowledge or Ready, has to be controlled by the break-point logic and FRZ. It has to insert wait states at the right phase, as specified by the processor manufacturer. With such a scheme, a non-invasive control of the target processor can be achieved.

5.7 *General Block Diagram of TSI*

There are a lot of common features between the different Target-Specific Interfaces, which can be utilized to speed up the design process of a TSI. Therefore, the next step is to give a general block diagram for TSI, which will separate out the common interface blocks and the processor-specific ones. The blocks common to all TSIs are the ID comparison block and the control I/O, data and address buffers to the target bus. A brief outline of the general block diagram is given below, with reference to Figure 5.1, which is self-explanatory.

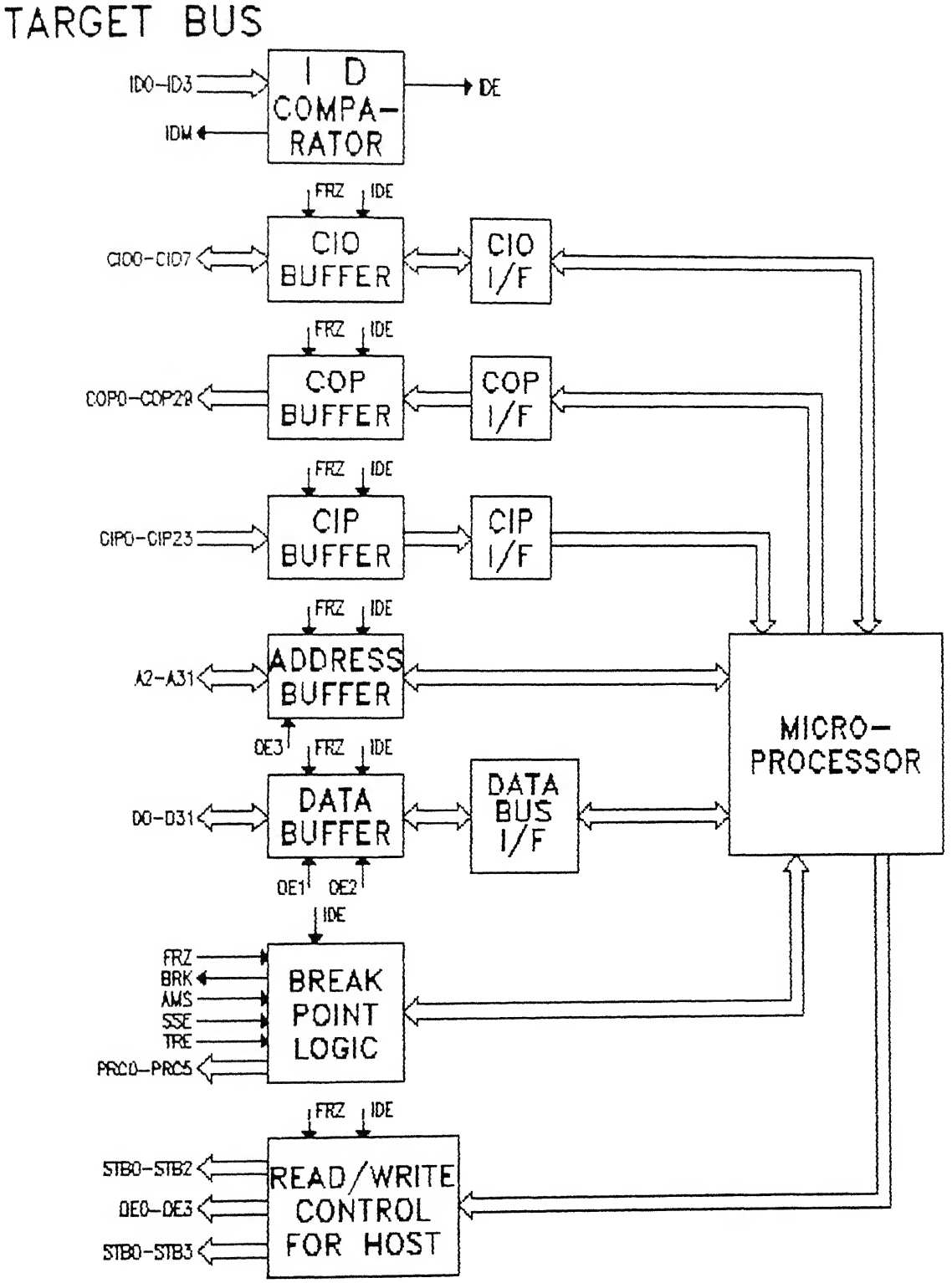


Figure 5.1: General Block Diageam of a TSI

The ID comparison block is independent of the processor and its signals, so it can be implemented as described in section 5.1, on any TSI. The processor control signals are conditioned, if required, in the CIP, COP and CIO interfaces. These blocks are connected to the target bus through the control I/O buffer. The COP and CIO signals are enabled by BEN, which is given by:

$$BEN = IDE \cdot \overline{FRZ}$$

BEN is used to latch the CIP inputs into a register also. The CIP register and COP buffer are unidirectional, while the CIO buffer is bidirectional, requiring a direction control. This is supplied by the OE0 signal from the Read/Write generation block, which controls the CIO register in the host cards. The address and data buffers are also enabled when the combination logic of BEN, given above, is satisfied. Both these buffers are bidirectional. The direction of the data buffer is determined by a combination of OE1 and OE2, when either one of them is asserted, a read operation is taking place on the target bus. Similarly, the direction of the address buffer is controlled by OE3. The data interface block is processor-specific, which can be used to implement swap logic for processors requiring it. In the following sections, we will examine TSI required for a Motorola microprocessor, the 68020.

5.8 *Salient features of the 68020 and its TSI*

The 68020 has 32-bit data and address buses. It has a 256-byte instruction cache organised as 64 long-word entries. It supports compatible 32-bit coprocessors on a common coprocessor interface. Its dynamic bus sizing capability supports 8-/16-/32-bit memories and peripherals, checking the size on a cycle to cycle basis. Each bus cycle

consists of 6 phases, S0-S5, requiring a minimum of three clock cycles. During S0, valid address, function code, R/$\overline{W}$ and size information is put on the bus by the processor. During S1, $\overline{AS}$ and $\overline{DS}$ are asserted. In the S2 phase, $\overline{DBEN}$ is asserted for external buffers, and the data put on the data bus for a write cycle. At the end of S2, the processor starts sampling the $\overline{DSACK0}$ and $\overline{DSACK1}$ lines. If both are negated, wait states are inserted. They are sampled on falling edges of the clock till the assertion of one of them is recognised. Data is latched at the end of S4 for a read cycle, and during S5, $\overline{AS}$ and $\overline{DS}$ are negated.

The Target-Specific Interface required for the 68020 conforms to the general block diagram described in the previous section. The 68020 signals are mapped as per the table given in Appendix C. The common blocks of ID comparison block, control I/O, address and data buffers are almost identical to the ones in the general block diagram. The differences are in the control I/O and address buffers. In the control I/O block, as the CIO lines are not used, the corresponding buffer need not be implemented and OE0 need not be connected. The address buffer can be unidirectional, and so this block will not require OE2 for direction control. The other blocks are specific to this TSI, and will be described in the following sections. Reference is made to the block diagram in Figure 5.2.

5.9 *Break-point Logic*

The break-point logic can detect six hardware break conditions for the 68020. They are page change, single-step execution over, CPU cycle, Bus float, Halt and Reset. The first three break conditions are related to the bus cycle and hence use the address strobe $\overline{AS}$ generated by the processor, to qualify their detection.

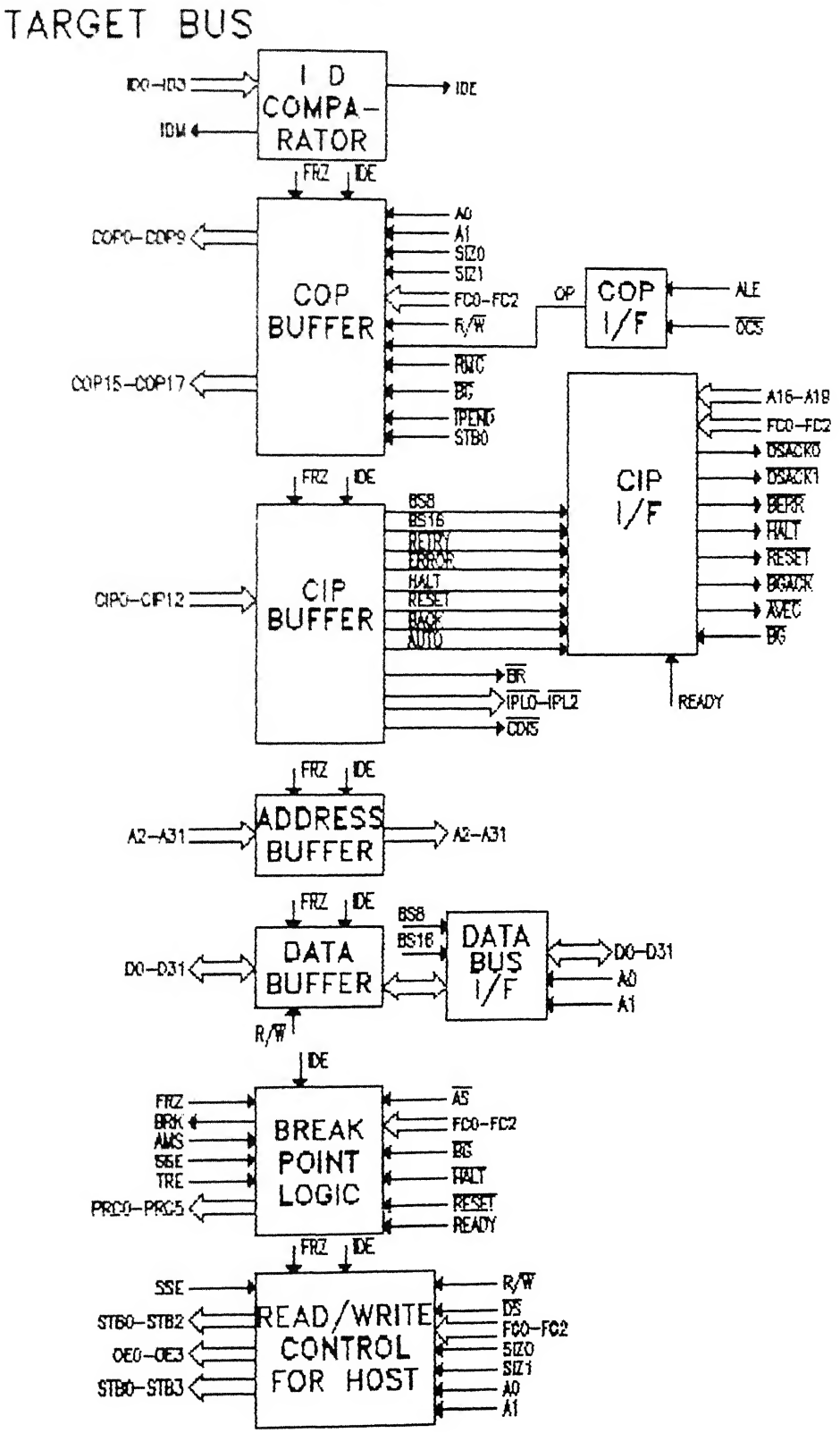


Figure 5.2: Block Diagram of TSI for 68020

Page change is detected by gating AMS from the host cards with $\overline{AS}$ to get the status of *page change* in the current cycle. *Single-step execution over* is asserted when the current bus cycle is completed and $\overline{AS}$ for the next cycle is asserted. It is gated by SSE, the enable signal from the host.

68020 defines three types of CPU space cycles - the interrupt acknowledge cycle, the break-point cycle and the coprocessor access cycle. The CPU space cycle is identified by high levels on all the FC lines. So, in the break-point logic, the *CPU cycle* detection is signalled when FC0-FC2 are all HIGH and $\overline{AS}$ is asserted. Individual types of CPU space cycles can be identified from the address bus.

The *bus float* condition is indicated when the $\overline{BG}$ output of the processor is asserted with $\overline{AS}$ negated. Halt and Reset detection are signalled by the respective lines from the processor.

The next step is to generate $\overline{DSACK0}$ and $\overline{DSACK1}$ to provide data acknowledge signals to the 68020, and BRK and PRC signals to the host cards. The $\overline{DSACK}$ signals acknowledge the data as well as the size of the data bus that can be handled. Therefore, the Ready signal generated in the break-point logic has to be gated with the BS8 and BS16 signals in the CIP Interface for generating $\overline{DSACK}$ signals to the processor. The Ready signal is negated if page change, single-step execution over or CPU cycle is detected or FRZ is asserted when $\overline{AS}$ is asserted. Care has to be taken to ensure that Ready is negated before the end of S2 phase in the current bus cycle, when such a condition is met.

BRK is asserted when at least one of the break conditions is detected. It is gated by FRZ and $\overline{AS}$ delayed to S5, to ensure that trace information for the current bus cycle is stored in the trace buffer. The six PRC signals reflect the six break-

point detection signals. BRK and PRC signals are connected to the target bus through buffers, which are enabled when IDE is asserted.

5.10 *Read/Write Controls to the Host Cards*

Since valid trace information is available during the S3 and S4 phases of the bus cycle, STB0 is asserted for this period. STB1 is the processor clock gated by FRZ, synchronised to the processor clock to prevent spikes on the output.

OE0, the read control for the CIO register in the host cards, need not be generated, since these lines are not required for this TSI. OE2, the read control for the data register in the host cards, has to be asserted for read operations on data memory during single-step execution. Therefore, OE2 can be implemented by the logic:

$$\overline{OE2} = R/\overline{W} \cdot FC0 \cdot \overline{FC1} \cdot \overline{DS} \cdot SSE$$

OE3, the read control for the address register, has to be negated when 68020 is being run, since its address bus is an output only.

STB2, OE1 and SB signals are controls for the local memory. OE1 has to be asserted whenever $\overline{DS}$ is asserted, $R/\overline{W}$ is high and OE2 is negated. STB2 has to be asserted when $\overline{DS}$ is asserted and $R/\overline{W}$ is low. The byte select signals are derived from SIZ1, SIZ0, A1 and A0 signals. The SIZ1 and SIZ0 signals indicate processor requirements of 1, 2, 3 and 4 bytes on the data bus for the combinations 01, 10, 11 and 00 respectively. SB3 has to be asserted for the A1-A0 combination of 00. the corresponding assertions are — SB2 for 01, SB1 for 10 and SB0 for 11. Additional SB signals have to be asserted depending on the data bus size required.

All the read and write controls are passed along to the target bus through buffers, which are enabled when IDE is asserted and FRZ is negated.

5.11 *Data Bus Interface*

The byte selection logic described in the previous section enables the host to simulate a 32-bit device on the processor bus directly. But to study the effect of 8-bit and 16-bit devices with the 68020, we have to simulate them on the target bus. The size of the bus acceptable to the host is programmed by it as two CIP lines, BS8 and BS16. These bits are used to gate the Ready signal from the break-point logic to the processor. The same bits are used in the data bus interface to implement a swap logic.

The 68020 expects a 16-bit device to be connected to the D16-D31 lines and an 8-bit device on the D24-D31 lines. Though the target bus accommodates 32-bit data, it has to be connected to the appropriate segment of the processor data bus corresponding to the programmed bus size. This is implemented by switching data bytes on the target bus to the required sections of the processor data bus through buffers. The enable signals to these buffers is generated from a logic based on A1, A0, BS8 and BS16. BS8 and BS16 determine the size of bus required. 32-bit devices are simulated by enabling the direct buffers on all bytes. 16-bit devices are simulated by enabling the direct buffers on D16-D31 if A1 is 0, otherwise D0-D15 buffers are enabled onto it. 8-bit devices are simulated by enabling the buffers switching the corresponding bytes to D24-D31, as decoded from A1 and A0. The direction control of these buffers is given by $R/\overline{W}$.

This swap logic takes care of simulating dynamically the various bus sizes in a processor environment. It allows the user to use the complete 256 x 32 local memory for his test code, independent of the programmed bus size.

5.12 *CIP and COP Interface*

These blocks take care of those CIP and COP signals which require conditioning before they can be utilized. BS8 and BS16, as programmed by the host, have to be gated with the Ready signal to generate $\overline{\text{DSACK0}}$ and $\overline{\text{DSACK1}}$ respectively. When any of the host-defined inputs ERROR or RETRY is asserted, $\overline{\text{BERR}}$ has to be asserted during S3 and S4 by gating these inputs with the Ready signal. $\overline{\text{HALT}}$ has to be asserted either when the Ready signal is asserted with RETRY asserted, or when HALT input from the CIP buffer is asserted. Both $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ require open-collector drivers for inputs, since they are bidirectional signals. The AUTO control input has to go as $\overline{\text{AVEC}}$ to the processor during an interrupt acknowledge cycle; this is accomplished by gating the AUTO signal with the FC0-FC2 and A16-A19 signals from the processor. The BACK control input is enabled to give the processor a $\overline{\text{BGACK}}$ signal only when $\overline{\text{BG}}$ from the processor is asserted.

The only processor output requiring conditioning is $\overline{\text{DCS}}$, which has to be stretched to the S5 phase of the bus cycle for it to be interfaced as a COP signal on the target bus. This is achieved by latching $\overline{\text{DCS}}$ by $\overline{\text{AS}}$ during each bus cycle.

Besides this circuitry, the 68020 requires an external clock generator. Since our interest is only a study of this processor, and not its emulation at full speed, we can choose the minimum possible frequency for the clock. All versions of the 68020 will operate with a 12.5 MHz clock, and hence this frequency can be used.

The design described above just illustrates one possible method of implementation of TSI for the 68020. It has not been assembled and tested. In the next chapter, a TSI will be described which has been used to test the host cards.

Chapter 6

TESTING OF THE GENERAL INTERFACE

After the implementation of the circuitry on the PC–host interface and host cards described in Chapters 3 and 4, the next step is to test them out. This could be done by connecting a suitable target processor and its interface to the target bus. But it should be possible to run the processor independently of the test system also, so that a step by step testing is possible. The best choice would therefore be an evaluation board or a development system.

6.1 *Choice of Test Setup*

The choice had to be made from two simple development systems available in the laboratory – one for the 8085A and the other for the 68000. The 8085A development system is a stand-alone system, with most of the signals from the 8085A processor available in a buffered form on a 44-pin edge connector for expansion boards. It has a designated memory area for the user, but permits use of the free space by external hardware. The RDY signal is wired-OR, permitting external hardware control. There is no time-out built into this system for continuation of bus cycles in the absence of RDY.

The SBC 68K is an evaluation board for the 16-bit 68000 processor. It requires an external host system or dumb terminal to operate it. This also has most of the processor signals available for external use on the expansion bus. It allocates memory area for the user, but permits expansion into the free space. The $\overline{\text{DTACK}}$ signal, which is equivalent to the RDY of the 8085A, can be controlled externally, but an internal time-out generator intervenes after 40 wait states to abort the bus cycle. This time is not sufficient for the host system to complete its operations on the host cards.

The 8085A is selected as it is a stand-alone system which does not abort the bus cycle if RDY is negated The only drawback is that it has an 8-bit data bus and a 16-bit address bus, which will not permit complete testing of the available resources of the host But, at this stage, we are interested in testing the mutual interfacing of the PC-AT, host cards, and the target processor through its TSI, rather than evaluating the total capabilities of the design. This can be done adequately using the 8085A development system.

Having chosen a particular development system for the purpose of testing the General Interface, we have to consider a suitable Target-Specific Interface (TSI) which will enable us to interconnect the development system with the test system. This TSI should be distinguished from an actual TSI to be used for the purpose of studying the microprocessor used in the development system.

6.2 *Design of TSI for Testing the Interface*

This TSI has been designed with most of the general features discussed in the previous chapter. However, since the development system has its own memory, which can be used

by the user for his own programmes, this design has not utilized the local memory of the host cards. The development system has its own assembler/disassembler and debug facilities which are available to the user and there is no need to duplicate it in the host at this point. A feature of permitting independent operation of the development system, when required by the user, has to be added. This is implemented by creating a mismatch in the ID. The host system can intervene into the execution of programmes on the processor and control it subsequently, as long as the ID put out by the host matches the processor ID.

With reference to the general block diagram described in section 5.7, the common circuitry of ID comparison and CIP register in the control I/O buffer has been implemented as described. The only difference is in the control of the CIP register. THe outputs of the CIP register are enabled onto the processor only if the ID on the target matches the processor ID. This permits independent operation of the development system as explained earlier. The CIO lines have not been utilized, and so its buffer and interface to the processor are not required. The address, data and COP buffers are not used, since only one target processor is being accessed and the local memory in the host is not being used. No special processor interface is required for the CIP and COP lines. Similarly, the data lines do not require any extra interface. The following sections describe the break-point logic and read/write logic which are processor-specific. Reference is made to the the block diagram in Figure 6.1. The allocation map of the signals from the development systems to the target bus is given in Appendix C. All the target bus signals have already been described in Chapter 4.

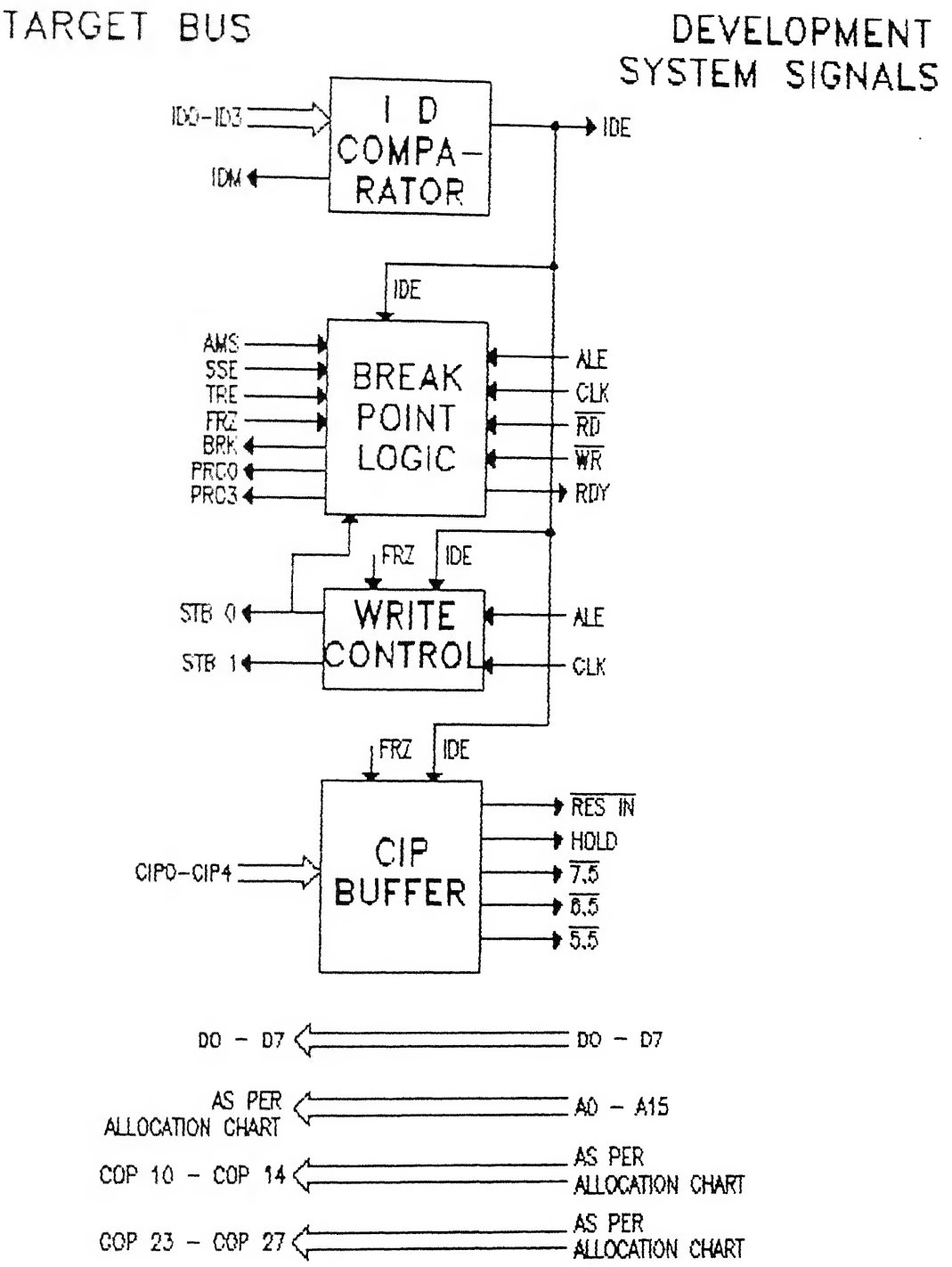


Figure 6.1: Block Diagram of TSI for 8085A Development System

6.3 *Break-point Logic*

Two break-point conditions have been implemented – page change and single-step execution over. The *page change* condition corresponds to a read/write cycle of the target processor occurring with AMS and TRE asserted and FRZ negated. It is latched when FRZ is asserted, to give PRC0. For sensing the *single step execution over* condition, ALE from the processor is used for the indication that the execution of the current bus cycle is over. It is gated by SSE and TRE and enabled when FRZ is negated. This signal is latched on the assertion of FRZ to give PRC3.

These two indications are combined to negate RDY if either condition is detected. The same combined signal is used to generate BRK when STB0 and FRZ are negated. The RDY and BRK signals are passed through a buffer enabled by IDE.

6.4 *Read and Write Controls for the Host*

Since the CIO, Address and Data Registers and the local memory in the host cards are not being used, their control signals are not being generated here. Only STB0 and STB1 have to be generated. STB0 is generated from ALE which is delayed to the T2 state using the processor clock. STB0 is asserted for one clock period during T2 state of the 8085A machine cycle, when valid data, address and bus-related control outputs are available. STB1 is the processor clock from the development system. Both signals are enabled when IDE is asserted and FRZ is negated.

This TSI has been implemented on a breadboard as this is not going to be a permanent feature of the test system. The following sections will give an idea of how to use this setup in capturing trace information from the 8085A processor, while

controlling execution of programmes on it and programming its control inputs from the host.

6.5 *Common Software Required for Operation of the Test System*

The test system requires three subroutines for using the PC-AT as the controller for its transactions with the host cards – for input handling, output handling and interrupt handling respectively. Unlike the input and output routines commonly available for PCs, these two routines have to utilize the complete 16-bit data bus of the PC-AT for access to the ports. Assembly-level subroutines for the 80286 have been written to take care of this requirement by using the IN and OUT instructions for 16-bit data.

IRQ9, the interrupt request from the host cards, is redirected to Interrupt 0A (hex) by the BIOS of the PC-AT. So the vector of the Interrupt Service Routine (ISR) has to be loaded at the location corresponding to this interrupt in the vector table, replacing the existing one. The original vector should be restored in the vector table at the end of the user programme. At the end of the ISR, End of Interrupt (EOI) is set in both the interrupt controllers of the PC-AT to ensure a proper exit from the ISR. The main part of the ISR uses the input and output subroutines to have access to the registers and FIFOs on the host cards.

6.6 *Setting up Break-points*

Three types of hardware break-points are used in this test setup. They are FIFOs full, page change and single-step execution over. The *FIFOs full* condition can be disabled

by asserting the FIFR signal during the run cycle. This will keep the FIFOs empty and thus FF will not be generated. However, trace information is lost. Single-step execution can be disabled by negating SSE. Both *single-step execution over* and *page change* break-points are disabled by negating TRE, irrespective of SSE.

If the user wants to observe just the trace information, without halting on page change, he can negate TRE and FIFR. However, the upper half of the address may not be correct, since only the last value is available in the address latch of the trace buffer. If single-step execution has to be observed, SSE and TRE have to be asserted and FIFR negated. The observation of single-instruction execution has to be implemented in software by performing multiple single-step operations, using disassembly of the executed code and the status bits S1 and S0 to determine the end of the instruction. Tracing a page of programme execution is done by asserting TRE only.

6.7 *Setting up of the Control Inputs*

$\overline{\text{RES IN}}$, HOLD, $\overline{7.5}$, $\overline{6.5}$ and $\overline{\overline{5.5}}$ to the development system can be set from the test system to take effect on the corresponding inputs to the target processor. The operation merely involves the setting of the corresponding bits in the CIP register before asserting RUN.

6.8 *Observation of the Control Outputs*

Observation on the bus cycle trace information of the processor is similar to that performed by an evaluation board or emulator. But by observation of the asynchronous

control outputs, we can correlate it with the bus cycle activity and get a wealth of information. The number of clock cycles required for each machine cycle, the time required for HLDA to be asserted after HOLD is asserted by the host, and the time required for the processor to activate RES OUT after RES IN is asserted, are some examples of the data obtained by such an observation.

A demonstration programme, TEST, has been prepared to give an idea of the capabilities of the test system that have been tested so far. Subsequent sections will illustrate its usage with examples. More features can be added to it through the software for the handling of the target processor.

6.9 *Description of the Menus in TEST*

TEST starts off with the display of *Opening Menu*, which offers the four options listed below. The user selects an option by entering its serial number.

1. Select new processor.
2. Display the allocation map of a processor.
3. Go to Run menu.
4. Exit.

The first option allows the user to select a new processor from those available in the library whose TSIs are connected to the target bus. If its TSI is not on the target bus, the choice is rejected and the previous selection holds good. The second option displays the allocation map of any processor from the library. The user has to enter the generic number of the processor for which he needs the map. The *Exit* option is the default option, and performs a clean exit from the demonstration programme, by restoring the original conditions in the PC-AT. All target processors are deselected

before exit to DOS. The third option leads the user to the *Run menu*, which is the main menu for running of the selected target processor.

The Run menu has 5 options

1. Go to Breakpoint Menu.
2. Go to CIF menu
3. Go to Display menu
4. Run
5. Exit to Opening Menu

The first option leads to the *Break-point sub-menu*, in which the user sets up the break-point conditions from the following available choice:

1. Page change
2. Single-step execution.
3. Continuous Trace.
4. Continuous run.

The default choice is page change. With this choice, the processor can be run till it requires data from a different page or till the trace buffer is full, whichever occurs earlier. With single-step execution, the processor can be run for the current bus cycle. In continuous trace, page change and single-step execution are disabled, so the processor can be run continuously until the user deliberately stops its execution. The last set of trace information is available in the trace buffer. The *Continuous run* is an additional option which permits a development system or an evaluation board, connected to the target bus, to be operated from its own keyboard independent of the host system whenever required. It is similar to continuous trace, except that the last trace information is not available.

The second option in the Run menu leads off to the *CIP sub-menu*, in which the user can programme the available control inputs, like Reset Out, Hold, RST7.5, RST6.5 and RST5.5 in the case of the 8085A development system. This is in the form of a query and response for each input.

The third option is the *Display sub-menu*. Two types of display formats are possible, as shown in Appendix E. They are Status and Trace. In the Status display, information about the cause of break, the last trace information in the case of a break due to page change or completion of single-step execution, and the status of each of the control outputs from the Clock cycle Trace Buffer (CTB) are displayed. The status of the control outputs from the CTB is given for the first clock cycle and the last clock cycle, along with any intermediate ones during which the state of the output changes. It is restricted to a maximum of two intermediate clock cycles. In the Trace display, the address, data, type of bus cycle and number of clock cycles required for its execution is displayed. This is done on a page by page basis, with an option to escape after seeing the current page on the screen. The Display menu allows the user to select one of the following:

1. Trace only.
2. Status only.
3. Trace and Status.

The fourth option in the Run menu runs the target processor till a break-point is reached, or the user stops execution, by pressing any key. After seeing the information from the target, he can return to the Run menu, or run the processor again with the same setup.

The last option is to return back to the Opening Menu. The format of these menus are given in Appendix E. Next, we will take up some specific examples of the operation of TEST. In each example, the italicized portions denote actual strings seen on the monitor of the PC-AT, they are either commands keyed in by the user or programme outputs.

6.10 *Selection of a new processor*

	USER COMMAND	PROGRAMME RESPONSE
1	TEST (cr)	Opening Menu is displayed. *Enter selection* (4)
2	1 (cr)	Screen is cleared. Current processor is displayed. Available processors in library and on target bus are also listed. Finally: *Enter new processor*
3	Enter Generic name of processor	Informs whether new choice is accepted. If not, the reason. Gives command: *Press q to quit*
4	q (cr)	Returns to Opening menu with command : *Enter selection* (4)

6.11 *Use of Page Change Break-point*

	USER COMMAND	PROGRAMME RESPONSE
1.	—	Displays Opening menu with command: *Enter selection* (4)

	USER COMMAND	PROGRAMME RESPONSE
2	3 (cr)	Page cleared and Run menu displayed with command: *Enter selection* (5)
3	1 (cr)	Break-point menu displayed with command: *Enter selection*
4	4 (cr)	Continuous run selected. Gives command: *Press q to quit*
5	q (cr)	Returns to Run menu with same command as in 2.
6	2 (cr)	Displays individual queries for enabling CIP lines.
7	Respond in each case with *n* (cr)	Returns to Run menu with same command as in 2.
8	3 (cr)	Display menu on screen with command : *Enter selection*
9	3 (cr)	Trace and status display selected. Gives command: *Press q to quit*
10	q (cr)	Run menu displayed with same command.
11	4 (cr)	Sets the processor to continuous run.
12	Programme given in Appendix E is entered in the development system and then a key is pressed on the PC-AT keyboard	Status is displayed on the screen, with command: *Press c to continue*
13.	q (cr)	Returns to Run menu with same command.
14	1 (cr)	Break-point menu displayed with its command.

	USER COMMAND	PROGRAMME RESPONSE
15	1 ⟨cr⟩	Page change selected,and quit command given.
16	q ⟨cr⟩	Returns to Run menu with same command.
17	4 ⟨cr⟩	Sets the processor to run till page changes. Then Status displayed with cause of break as *page change* and current cycle trace information. Gives command: *Press c to continue*
18	q ⟨cr⟩	Returns to Run menu with command.
19	5 ⟨cr⟩	Returns to Opening menu.

6.12 *Single-step Execution*

Steps 1 to 13 of section 6.11 are to be repeated, if the user has got out of TEST or changed any other setup parameters. Otherwise he can proceed directly in the following manner.

	USER COMMAND	PROGRAMME RESPONSE
1.	1 ⟨cr⟩	Break-point menu is displayed with its command.
2.	2 ⟨cr⟩	*Single-step execution over* is selected, with quit command.
3.	q ⟨cr⟩	Run menu displayed with its command.
4.	4 ⟨cr⟩	Processor is run for one bus-cycle. Then a continue command is given.
5.	c ⟨cr⟩	Response as in 4.

Steps 4 and 5 can be repeated for each step to be traced. Finally, instead of the user command as in 5, steps 6 onwards are followed.

6	*q* ⟨*cr*⟩	Run menu displayed with its command.
7	5 ⟨*cr*⟩	Goes back to the opening menu.

6.13 *DMA Controls*

	USER COMMAND	PROGRAMME RESPONSE
1	—	Displays Opening menu with command : *Enter selection* (4)
2	3 ⟨*cr*⟩	Page cleared and Run menu displayed with command: *Enter selection* (5)
3	1 ⟨*cr*⟩	Break-point menu displayed with command: *Enter selection*
4.	1 ⟨*cr*⟩	Page change break-point selected. Gives command: *Press q to quit*
5.	*q* ⟨*cr*⟩	Returns to Run menu with same command as in 2.
6.	2 ⟨*cr*⟩	Displays individual queries for enabling CIP lines.
7.	Respond in each case with *n* ⟨*cr*⟩ except for HOLD; response for it should be: *y* ⟨*cr*⟩	Returns to Run menu with same command as in 2.
8.	3 ⟨*cr*⟩	Display menu on screen with command : *Enter selection*

	USER COMMAND	PROGRAMME RESPONSE
9	2 (cr)	Status display selected. Gives command: *Press q to quit*
10	q (cr)	Run menu displayed with same command.
11	4 (cr)	Runs the processor till page break or buffer full. Then Status displayed, with clock cycle when HLDA goes low.

Repeat steps 5 to 11 with HOLD also disabled in step 7 and single-step selected in Break-point menu. In 11, HLDA would have become inactive.

6.14 *Conclusions*

The demonstration programme gives a reasonable idea of the capabilities of the Test System and how it can be used effectively. However, some of the important facilities are not demonstrated, like the use of the local memory in loading programmes to be run. The use of local memory is not tested, since a development system is used in place of the target processor with its existing environment. By taking up the development of the Target-Specific Interface for a 32-bit microprocessor like the Motorola 68020, the complete capabilities of the system can be tested. Also, the software has to be written with library functions for as many processors as possible, so that a fast, interactive programme is created for the user. These are the two major fields of activity for future work on this system.

BIBLIOGRAPHY

JOURNALS

1. Freid, S. "Accessing Hardware from 80386 Protected Mode Part I," Dr. Dobb's Journal, pp 92 -96, May 1990.
2. Cooper, T.C., et al, "A Benchmark Comparison of 32-bit Microprocessors," IEEE Micro, pp 53 - 58, August 1986.
3. Lawday G., "Emulate or Speculate ?," Electronics World + Wireless World, pp 126 - 129, February 1991.
4. Vasko, D.A and Weyer, D., "Simulation Sorts out System Schemes by Surveying Trade-offs,", Electronic Design, pp 85 - 93, September 27, 1990.

BOOKS

1. Kernighan, B.W. and Ritchie, D.M., "The C Programming Language," Second Edition, Prentice-Hall of India Private Ltd., 1990.
2. Murray, W.H. and Pappas, C.H., "80386/80286 Assembly Language Programming," Osborne.
3. Krutz, R.L., "Interfacing Techniques in Digital Design with Emphasis on Microprocessors," John Wiley and Sons, 1988.
4. Rajaraman, V. and Radhakrishnan, T., "An Introduction to Digital Computer Design," Third Edition, Prentice-Hall of India Private Ltd., 1988.
5. Wiatrowski, C.A. and House, C.H., "Logic Circuits and Microcomputer Systems," McGraw-Hill International Book Company, 1980.

6 Nicoud, J.D., and Wagner, F., "Major Microprocessors: A Unified Approach using CALM" North-Holland Private Co., 1987

7 Clements, Alan, "Microprocessor Interfacing and the 68000 Peripherals and Systems" John Wiley and Sons 1989

8 Auslander, A.M. and Tham, C.H., "Real Time Software Control Program Examples in 'C'" Prentice-Hall 1990

MANUALS

1 i486 Microprocessor Manual. Intel Corporation, April 1989.

2 Microprocessors Data Book Vol II, Intel Corporation, 1991.

3 MC68000 Family Reference Manual, M68000 FR/AD, Motorola Inc., 1990.

4 Static RAMs Data Book, First Edition, SGS-Thomson Microelectronics, November 1990.

5 SBC 68K User's Manual, Revision 1.1, 1 Edition, Arnwesh Inc., 1990.

6 Work Station User's Manual, Third Issue, Version 4.0, Dept. of Electrical Engineering, I.I.T. Kanpur, September 1986.

7 VME Bus Specification Manual, Revision C, Philips, February 1985.

8 Technical Reference Personal Computer AT, Personal Computer Hardware Reference Library

Appendix A

COMPARATIVE STUDY OF MICROPROCESSORS

For the purpose of allocating signals between the General Interface and Target-Specific Interface as discussed in Chapter 2, we have compared four CISC microprocessors — 68030 and 68040 from Motorola, and 80386 and i486 from Intel. All these are 32-bit processors. The 68030 and 80386 representing previous generations of microprocessors, while the other two are the latest and most complex among the processors available today. This study does a comparison of common features of signal handling between the processors, and does not compare their architectures or performance. The signals have been grouped according to their functions, and then compared in Tables A.1 to A.7, which are given in the following pages. Table A.8 compares the number of control lines required by each processor.The tables are listed below.

Table A.1 — Asynchronous Bus Transfer

Table A.2 — Burst Transfer

Table A.3 — Cache Control

Table A.4 — Interrupts

Table A.5 — Reset and Halt

Table A.6 — Bus Arbitration and DMA Control

Table A.7 — Other Functions

Table A.8 — Comparison of Control-Line Requirements

TABLE A.1 ASYNCHRONOUS BUS TRANSFER

CHARACTER-ISTICS	68030	68040	80386	i486
Address	A0–A31 (O)	A0–A31 (I/O)	A2–A31 (O)	A2–A3 (O) A4–A31 (I/O)
Data (I/O)	D0–D31	D0–D31	D0–D31	D0–D31
Function Control	FC0–FC2 (O) & R/$\overline{W}$ (O)	TT0–TT1 (I/O) TM0–TM2 (O) & R/$\overline{W}$ (I/O)	M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$ (O)	M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$ (O)
Byte Selection	Decode from SIZ0, SIZ1 (O) & A0, A1	Decode from SIZ0, SIZ1 (I/O) & A0, A1	$\overline{BE0}$–$\overline{BE3}$ (O)	$\overline{BE0}$–$\overline{BE3}$ (O)
Size Control	$\overline{DSACK0}$, $\overline{DSACK1}$ (I)	–	$\overline{BS16}$ (I)	$\overline{BS8}$, $\overline{BS16}$ (I)
Start	$\overline{AS}$, $\overline{OCS}$, $\overline{ECS}$ (O)	$\overline{TS}$ (I/O)	$\overline{ADS}$ (O)	$\overline{ADS}$
End	$\overline{DSACK0}$, $\overline{DSACK1}$	$\overline{TA}$ (I/O)	$\overline{RDY}$ (O)	$\overline{RDY}$ (O)
Error Condition	$\overline{BERR}$ (I)	$\overline{TEA}$ (I)	–	–
Min. clock cycles reqd.	3	2	4	2
Additional Features	Retry is possible	Bus Transfer, Control & Addr. are bidir. Status lines $\overline{TIP}$, $\overline{UPA1}$ & $\overline{UPA0}$	Addr. pipelining by $\overline{NA}$ (I) allows more access time for ext. device.	DP0-DP3 (I/O) & $\overline{PCHK}$ (I/O) for parity check.

TABLE A.2 BURST TRANSFER

CHARACTER-ISTICS	68030	68040	80386	i486
Burst Request	$\overline{\text{CBREQ}}$ (O)	–	Burst Mode not Available.	–
Burst Ack	$\overline{\text{STERM}}$ (I)	$\overline{\text{TA}}$ with $\overline{\text{TBI}}$ (I) negated.		$\overline{\text{BRDY}}$ (I) with $\overline{\text{RDY}}$ negated.
Burst Inhibit	$\overline{\text{STERM}}$ negated.	$\overline{\text{TBI}}$ asserted.		$\overline{\text{RDY}}$ assert.
No. of clk bet. transfers	1	1		1
Status and other controls	–	$\overline{\text{DLE}}$ (O) - data latch enable		$\overline{\text{BLAST}}$ (O) - last burst

TABLE A.3 CACHE CONTROL

CHARACTER-ISTICS	68030	68040	80386	i486
Instruction Cache size	256 bytes	4 Kbytes	No cache.	Combined with Data cache. 8 Kbytes Total Cache
Data Cache size	256 bytes	4 Kbytes	No cache.	
Organisation of Cache	16 x 16-byte lines, direct	64 sets x 4 way		128 sets x 4 way
Disable	$\overline{\text{CIIN}}$, $\overline{\text{CDIS}}$ (I)	$\overline{\text{TCI}}$, $\overline{\text{CDIS}}$ (I)		$\overline{\text{KEN}}$ (I)
Burst Ack	$\overline{\text{CBACK}}$ (I)	–		–
Status and Additional Controls	$\overline{\text{CIOUT}}$ (O) - CI bit status	$\overline{\text{TLN1}}$, $\overline{\text{TLN0}}$ (O) line no. being accessed		PWT, PCD (O) Page Status $\overline{\text{FLUSH}}$ (O) - cache flush.
Bus snooping and cache invalidation	–	SC1, SC0 (I) - Snoop Control $\overline{\text{MI}}$ (I) - snoops on ext. bus cycle to maintain cache coherency.		$\overline{\text{AHOLD}}$ (I) - ext. device invalidates cache at given addr.

TABLE A.4 INTERRUPTS

CHARACTER-ISTICS	68030	68040	80386	i486
Int Request	IPL0-IPL2 (I)	IPL0-IPL2 (I)	INTR (I)	INTR (I)
Int Vector	on data lines during ack cycle	on TM0-TM2 (I) during ack cycle	on D0-D7 during ack cycle	on D0-D7 during ack
Auto vector Inputs	$\overline{AVEC}$ - requests for int. vector	$\overline{AVEC}$ - requests for int. vector	NMI - nonmask interrupt	NMI -nonmask interrupt
Priority Level of requested interrupt	IPL0-IPL2	IPL0-IPL2	Not available	Not available
Status	$\overline{IPEND}$ (O)	$\overline{IPEND}$ (O)	–	–
Ack Cycle	async. bus read	async. bus read	async. bus read	async bus read

TABLE A.5 RESET AND HALT

CHARACTER-ISTICS	68030	68040	80386	i486
Reset In	$\overline{RESET}$ (I/O)	$\overline{RSTI}$ (I)	$\overline{RESET}$ (I)	$\overline{RESET}$ (I)
Reset Out	$\overline{RESET}$	$\overline{RSTO}$ (O)	–	–
Reset Instruction	Yes	Yes	No	No
Halt	$\overline{HALT}$ (I)	–	HLT Instruction	HLT Instruction
Other features	$\overline{HALT}$ & $\overline{BERR}$ asserted causes retry.	–	–	–

TABLE A.6 BUS ARBITRATION & DMA CONTROL

CHARACTER-ISTICS	68030	68040	80386	i486
Location of Bus Arbiter	Internal	External	Internal	Internal
Bus Request	$\overline{BR}$ (I)	$\overline{BR}$ (O)	HOLD (I)	HOLD (I)
Bus Grant	$\overline{BG}$ (O)	$\overline{BG}$ (I)	HLDA (O)	HLDA (O)
Bus Grant Ack	$\overline{BGACK}$ (I)	$\overline{BB}$ (O)	–	–
Other controls and status	–	$\overline{LOCK}$, $\overline{LOCKE}$ (O) for RMW cycles	$\overline{LOCK}$ (O) - for RMW cycles	$\overline{LOCK}$, $\overline{PLOCK}$ (O) for RMW $\overline{BOFF}$ (I) for immediate control to ext. device

TABLE A.7 OTHER CHARACTERISTICS

CHARACTER-ISTICS	68030	68040	80386	i486
Test Facility	Not Available	I/F for IEEE 1149.1 with TCK, TMS, TDI, TRST (all inputs) & TDO (O)	Self Test by $\overline{BUSY}$ LOW when $\overline{RESET}$ asserted	Self Test by AHOLD HIGH for two CLKs during $\overline{RESET}$ assertion
Mode of I/F for Coprocessor	Through CPU space bus cycle	–	H/W inputs $\overline{BUSY}$ $\overline{ERROR}$, $\overline{PEREQ}$	–
Emulator Support	$\overline{MMUDIS}$ (I) - disables MMU. $\overline{REFILL}$, $\overline{STATUS}$ (O) - status of pipeline.	$\overline{MDIS}$ (I) - disables MMU. PST0-PST3 (O) - status of int. execution	No ext. lines.	No ext. lines.
Debug Features	S/W Breakpt. thru' BKPT instruction & Brkpt. bus cycle	S/W Breakpt. thru' BKPT instruction & Brkpt. bus cycle	S/W Breakpt. for code & data thru' debug registers. Single-Step by setting TF	S/W Breakpt. for both thru' debug registers. Set TF to Single-Step

TABLE A.8 COMPARISON OF CONTROL-LINE REQUIREMENTS

CHARACTER-ISTICS	68030	68040	80386	i486
Control Inputs	16	18	9	13
Control Outputs —Bus related	12	11	8	13
Control Outputs —Asynchronous	4	9	2	2
Control I/O	—	7	—	4

Appendix B

DESCRIPTION OF SIGNALS ON BUSES

TABLE B.1 SIGNALS TO THE PC-HOST I/F CARD FROM THE PC-AT BUS

NAME	BUS PIN NO.	DIR.	ACTIVE LEVEL	BRIEF DESCRIPTION
SA0–SA19	A31 – A12	I/O	–	Address bits 0–19,gated when BALE is HIGH, and latched on its falling edge.
SD0–SD7	A9 - A2	I/O	–	Lower byte of data for 16-bit devices and only data path for 8-bit devices.
SD8–SD15	C11 – C18	I/O	–	Upper byte of data for 16-bit devices.
IOR	B14	I/O	LOW	I/O Read instructs an I/O device to drive its data onto the data bus.
IOW	B13	I/O	LOW	I/O Write instructs an I/O device to read the data on the data bus.
IRQ3–IRQ7 IRQ9 IRQ10–IRQ12 IRQ14, IRQ15	B25 – B21 B4 D3 – D5 D7, D6	ALL I	HIGH	They are used to signal to the system processor that an I/O device requires attention. IRQ9-IRQ15 has higher priority than IRQ3-IRQ7, IRQ9 has highest and IRQ7 the least priority.
RESET DRV	B2	O	HIGH	It is used to reset or initialise system logic at power-up time or during low line-voltage outage.
I/O CS16	D2	I	LOW	It signals the system board that the present transfer is a 16-bit, 1 wait-state I/O cycle. It should be driven by an open collector or tristate driver.
AEN	A11	O	HIGH	It signals that the DMA controller is in control of address & data buses, and all read/write commands.
+5 Vdc	B3,B29,D16	–	–	Power line.
GND	B1,B10,B31, D18	–	–	Ground.

NOTE: In column DIR., the direction of signals is given with respect to the system card of the PC-AT.

TABLE B.2 SIGNALS ON THE EXTENDED PC BUS

NAME	I/F CARD D CONNECTOR	MOTHER BOARD CONNECTORS	DIR.	ACTIVE LEVEL	BRIEF DESCRIPTION
DATA0– DATA7	13,25,12,24, 11,23,10,22	J2 – C17 - C24	I/O	–	Buffered Lower half of PC-AT Data Bus
DATA8– DATA15	1,14, 2,15, 3,16, 4,17	J2 – C25 - C32	I/O	–	Buffered Upper half of PC-AT Data Bus
ADD1 – ADD4	21, 9, 20, 8	J2 – C8 - C11	O	–	Buffered SA1-SA4 bits of PC-AT Address Bus
BIOR	5	J2 – C14	O	LOW	Buffered IOR signal.
BIOW	6	J2 – C15	O	LOW	Buffered IOW signal.
PROTO	7	J2 – C16	O	LOW	Decoded SA5-SA9 signal indicating prototype area selection.
RESET	19	J2 – B15	O	HIGH	Buffered RESET DRV from PC-AT bus
IRQ	18	J2 – B16	I	LOW	Interrupt Request from Host cards
GND	GND terminal	J1 – A9,A11,A15, A17,A19,B20,B23, C9 & J2 – B2,B12,B22,B31	–	–	Signal Ground

NOTE : The direction of signals given in DIR. column is with respect to the PC-Host Interface card.

TABLE B.3 SIGNALS ON THE TARGET BUS

NAME	SOURCE	ACTIVE LEVEL	MOTHERBOARD PIN NO.	BRIEF DESCRIPTION
FRZ	HOST	HIGH	J2–B19	Freezes target processor activity.
BRK	TSI	HIGH	J2–B20	Signals occurrence of break.
PRC0–PRC5	TSI	LOW or CODED	J1–A13,A14,C14, C12,A12,C13	Cause of break.
ID0–ID3	HOST	–	J1– C21, A21, C22, A22	Identification code for required processor.
IDM	TSI	LOW	J2– B18	Signals match of ID code.
AMS	HOST	LOW	J1– B18	Signals address doesn't match tag.
SSE	HOST	HIGH	J1– C16	Single-step execution enabled.
TRE	HOST	HIGH	J1– A16	All breakpts. in TSI enabled.
STB0	TSI	LOW	J1– C11	Write pulse for BTB.
STB1	TSI	LOW	J1– C23	Write pulse for CTB.
STB2	*	LOW	J1– A23	Write pulse for local memory.
OE0	TSI	LOW	J2– B3	Enable pulse for CIO register.
OE1	*	LOW	J2– B4	Read pulse for local memory.
OE2	*	LOW	J2– B23	Enable pulse for Data Register.
OE3	*	LOW	J2– B9	Enable pulse for Address Register.
SB0 – SB3	*	LOW	J1– C10, A10, C18, A18	Byte enable for local memory.

NOTE : * – These signals are driven by TSI during a run cycle, otherwise by the host cards.

NAME	SOURCE	ACTIVE LEVEL	MOTHERBOARD PIN NO	BRIEF DESCRIPTION
D0 – D15	I/O	–	J1 – C1, A1, C2, A2, C3, A3, C4, A4, C5, A5, C6, A6, C7, A7, C8, A8	Target data bus – lower *word*
D16 – D31	I/O	–	J1 – C24,A24,C25,A25, C26,A26,C27,A27, C28,A28,C29,A29, C30,A30,C31,A31	Target data bus – upper *word*
A2 – A15	I/O	–	J1 – C15,B15, C17,B17,C19,B19, C20,A20,B10,B11, B12,B13,B14,B16	Target address bus – lower *word*
A16 – A31	I/O	–	J1 – B1–B8 & B24–B31	Target address bus – upper *word*
COP0 : COP14	TSI	–	J2 – C3,A3,C4,A4, C5,A5,C6,A6, & B5–B11	Bus-related Control Outputs from Target
COP15 : COP24	TSI	–	J2 – A7–A11, A14–A16 & B14, B15	Asynchronous Control Outputs from Target
COP25 : COP29	TSI	–	J3 – 1–5	Asynchronous Control Outputs from Target
CIP0 : CIP15	HOST	–	J2 – A23,A21,A19,A17, A18,A20,A22,A24, A31,A29,A27,A25, A26,A28,A30,A32	Control Inputs to Target
CIP16 : CIP23	HOST	–	J2 – B23–B30	Control Inputs to Target
CIO0 : CIO7	I/O	–	J2 – C1,A1,C2,A2, C12,A12,C13,A13	Bi-directional Control signals of Target

Appendix C

ALLOCATION CHARTS

TABLE C.1 ALLOCATION CHART FOR HOST INPUT PORTS

PORT ADDR. Hex	NAME	RD CON. OE	WR CON. STB	D15–D12	D11–D8	D7–D4	D3–D0
300	STATUS	10	–	NOT USED	NOT USED	IDM,PRC5	PRC0,FF
302	ADDR.	11	0	A15	A10 0 0	NOT USED	NOT USED
304	ADDR.	12	0	A31			A16
306	DATA	13	0	D15			D0
308	DATA	14	0	D31			D16
30A	CIO/ ADDR.	15	0	CIO7	CIO0	A9	A2
30C	COP	16	0	EF0 COP14			COP0
30E	COP	17	1	EF1 COP29			COP15

TABLE C.2 ALLOCATION CHART FOR HOST OUTPUT PORTS

PORT ADDR. Hex	NAME	RD CON. OE	WR CON. STB	D15–D12	D11–D8	D7–D4	D3–D0
300	COMM- AND	–	10	NOT USED	NOT USED	RUN,FIFR,TRE, SSE	ID3----ID0
302	ADDR.	3	11	A15			A2 X X
304	ADDR.	3	12	A31			A16
306	TAG	–	13	TA15	TA10 0 0	NOT USED	NOT USED
308	TAG	–	14	TA31			TA16
30A	DATA	2	15	D15			D0
30C	DATA	2	16	D31			D16
30E	CIP	–	17	CIP15			CIP0
31X	CIO/ CIP	0*	18	CIO7	CIO0	CIP23	CIP16

NOTE: 1. The Read Control (RD CON.) OE0 for output port 31X is used for the CIO register only. The CIP half is permanently enabled.

2. – in OE column signifies the port is permanently enabled. In STB column it implies that no WRite COntrol is required.

TABLE C.3 ALLOCATION CHART FOR 68020

CONTROL OUTPUTS

COP7	COP6	COP5	COP4	COP3	COP2	COP1	COP0
R/$\overline{W}$	FC2	FC1	FC0	SIZ1	SIZ0	A1	A0

EF0	COP14	COP13	COP12	COP11	COP10	COP9	COP8
–	X	X	X	X	X	$\overline{RMC}$	OP

COP22	COP21	COP20	COP19	COP18	COP17	COP16	COP15
X	X	X	X	X	STB0	$\overline{IPEND}$	$\overline{BG}$

EF1	COP29	COP28	COP27	COP26	COP25	COP24	COP23
–	X	X	X	X	X	X	X

CONTROL INPUTS

CIP7	CIP6	CIP5	CIP4	CIP3	CIP2	CIP1	CIP0
BACK	$\overline{BR}$	RESET	HALT	ERROR	RETRY	BS16	BS8

CIP15	CIP14	CIP13	CIP12	CIP11	CIP10	CIP9	CIP8
X	X	X	$\overline{CDIS}$	AUTO	$\overline{IPL2}$	$\overline{IPL1}$	$\overline{IPL0}$

CIP23	CIP22	CIP21	CIP20	CIP19	CIP18	CIP17	CIP16
X	X	X	X	X	X	X	X

CONTROL I/O

CIO7	CIO6	CIO5	CIO4	CIO3	CIO2	CIO1	CIO0
X	X	X	X	X	X	X	X

STATUS

IDM	PRC5	PRC4	PRC3	PRC2	PRC1	PRC0	FF
–	Single Step Over	Reset	Halt	Bus Float	CPU Cycle	Page Change	–

TABLE C.4 ALLOCATION CHART FOR 8085A DEVELOPMENT SYSTEM

CONTROL OUTPUTS

COP7	COP6	COP5	COP4	COP3	COP2	COP1	COP0
X	X	X	X	X	X	X	X

EF0	COP14	COP13	COP12	COP11	COP10	COP9	COP8
–	IO/$\overline{M}$	S1	S0	$\overline{WR}$	$\overline{RD}$	A9	A8

COP22	COP21	COP20	COP19	COP18	COP17	COP16	COP15
X	X	X	X	X	X	X	X

EF1	COP29	COP28	COP27	COP26	COP25	COP24	COP23
–	X	X	S1	S0	STB0	HLDA	RES OUT

CONTROL INPUTS

CIP7	CIP6	CIP5	CIP4	CIP3	CIP2	CIP1	CIP0
X	X	X	RST7.5	RST6.5	RST5.5	RES IN	HOLD

CIP15	CIP14	CIP13	CIP12	CIP11	CIP10	CIP9	CIP8
X	X	X	X	X	X	X	X

CIP23	CIP22	CIP21	CIP20	CIP19	CIP18	CIP17	CIP16
X	X	X	X	X	X	X	X

CONTROL I/O

CIO7	CIO6	CIO5	CIO4	CIO3	CIO2	CIO1	CIO0
X	X	X	X	X	X	X	X

STATUS

IDM	PRC5	PRC4	PRC3	PRC2	PRC1	PRC0	FF
–	X	X	Single-Step Over	X	X	Page Change	–

NOTE : Address Map of Target Address Bus is :
A0–A7 is mapped to A2–A9 of Host; A10–A15 to A10–A15;
A8 to A16 & A9 to A18 of Host.

Data bits D0–D7 only are used as 8085A is an 8-bit processor.

Appendix D

SCHEMATIC DIAGRAMS

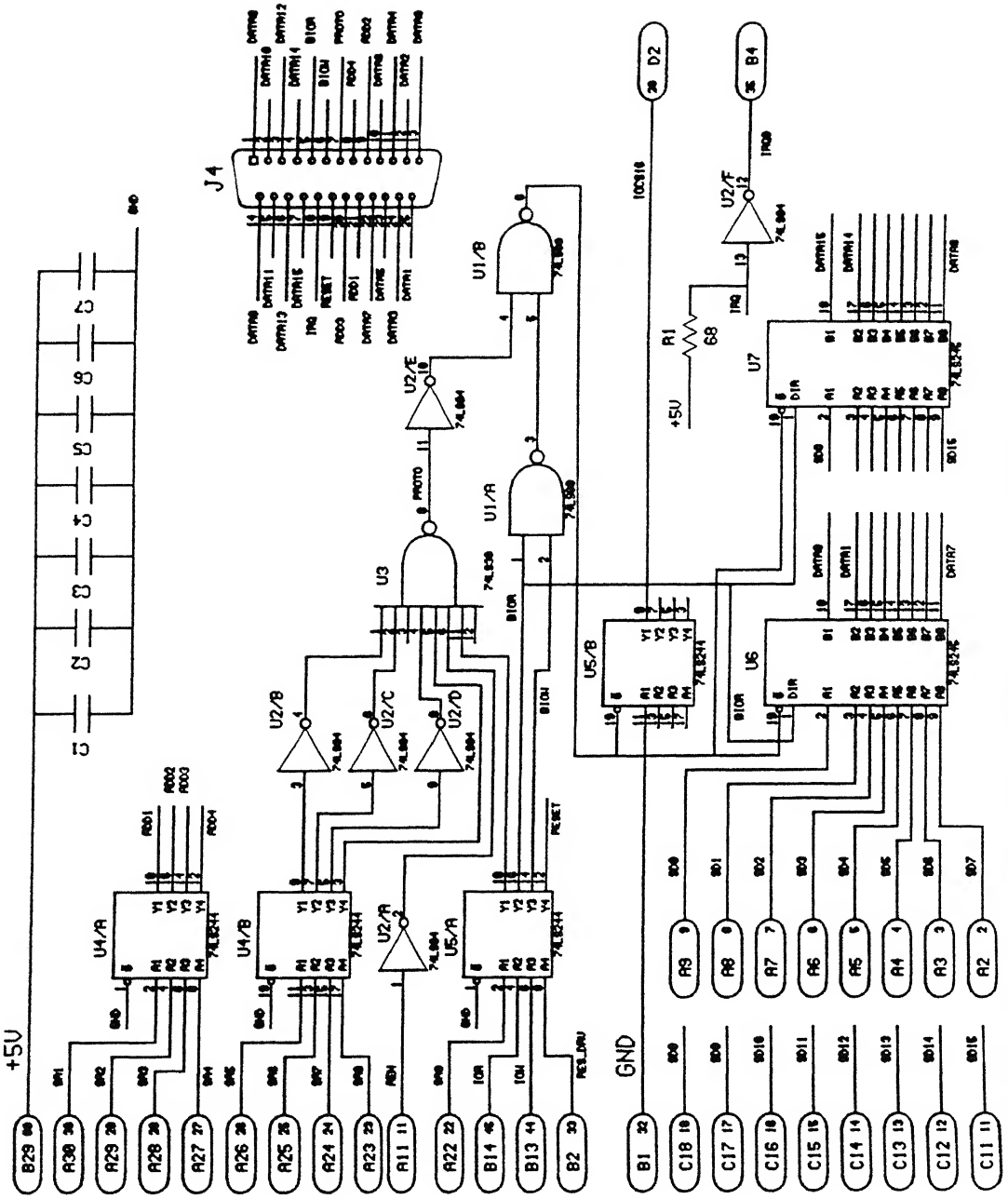


FIGURE D.1 PC-AT HOST I/F CARD SHT 1 of 1

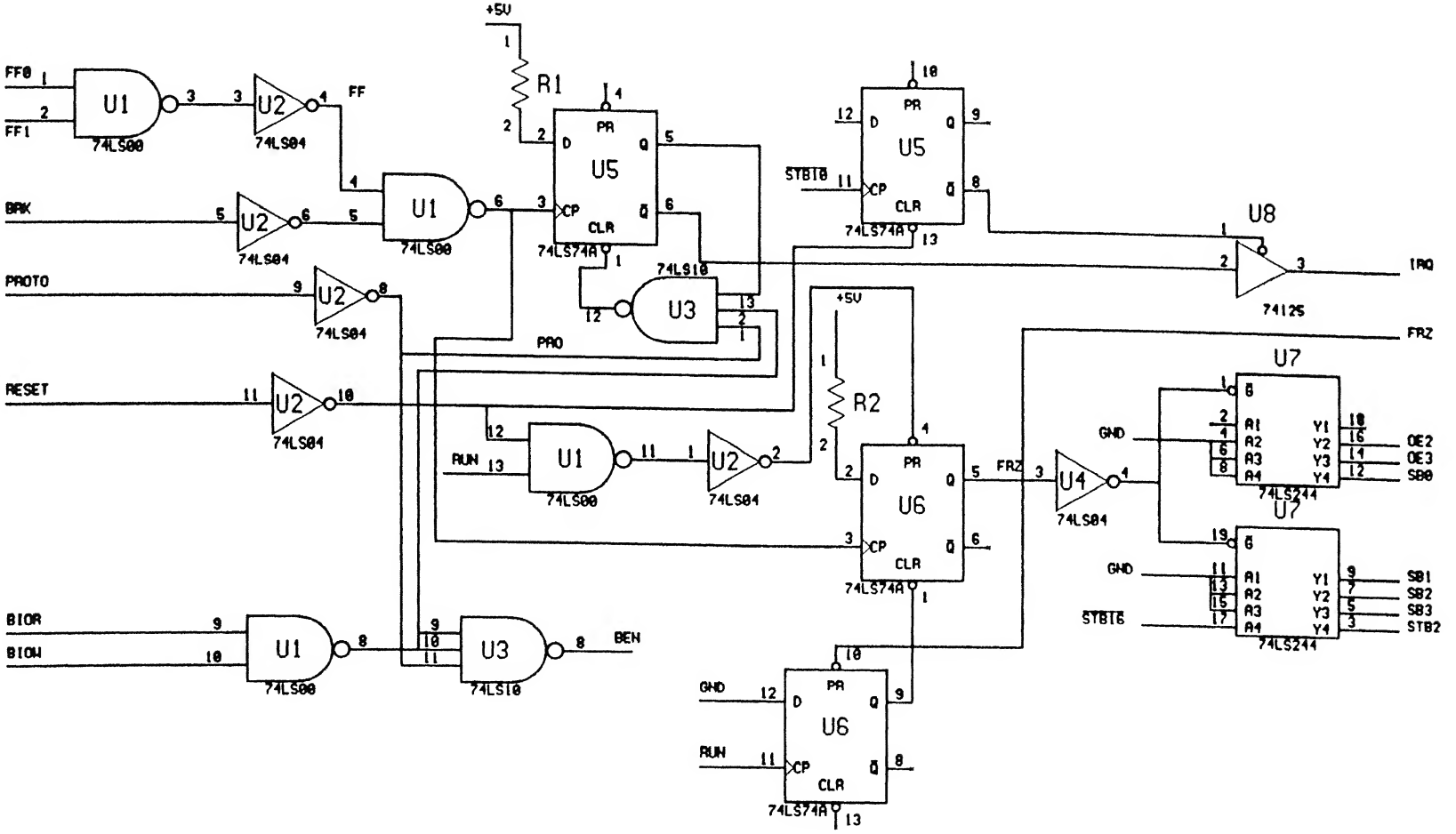


FIGURE D.2 HOST1 CARD SHT 1 of 5

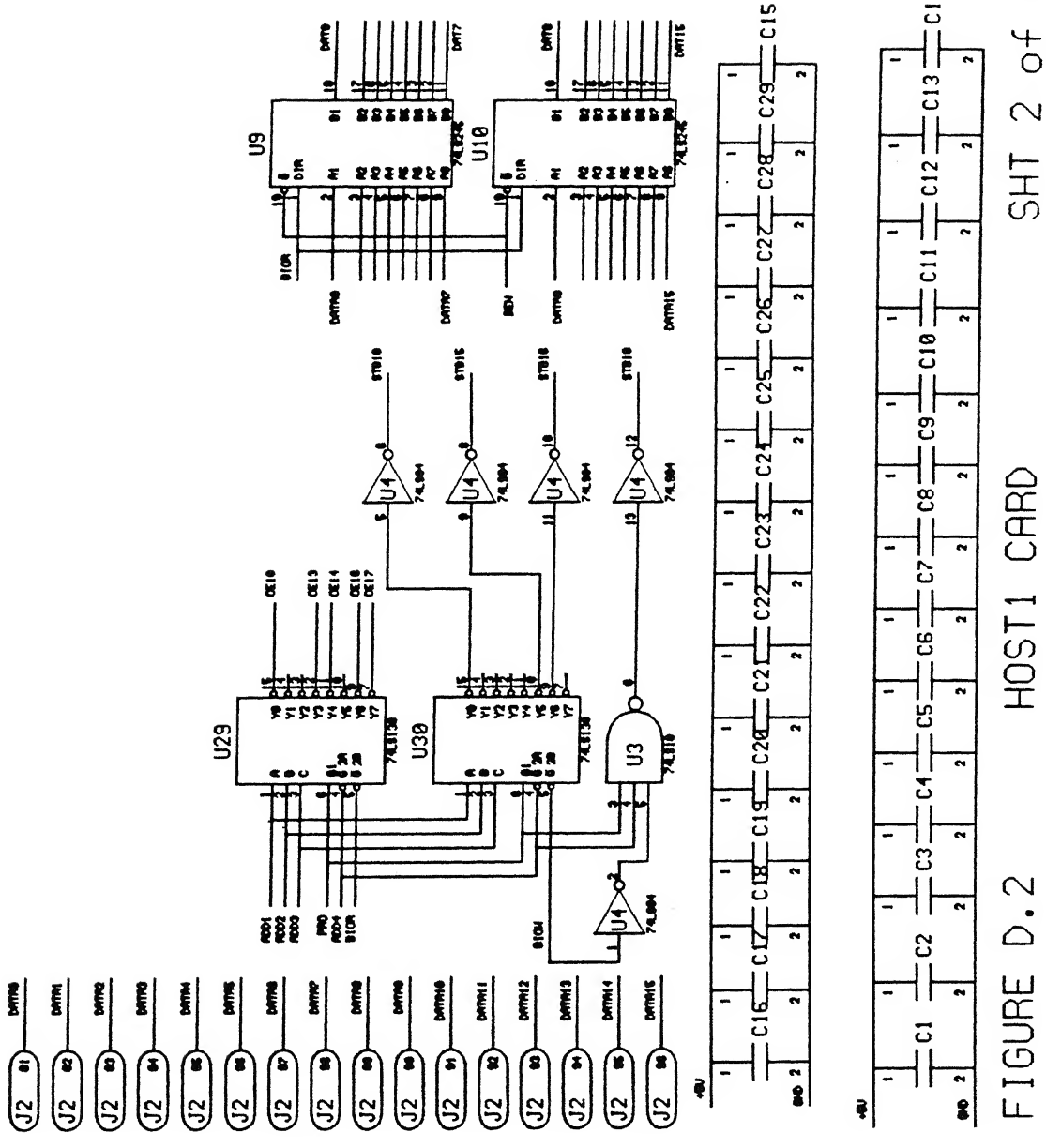


FIGURE D.2 HOST1 CARD SHT 2 of 5

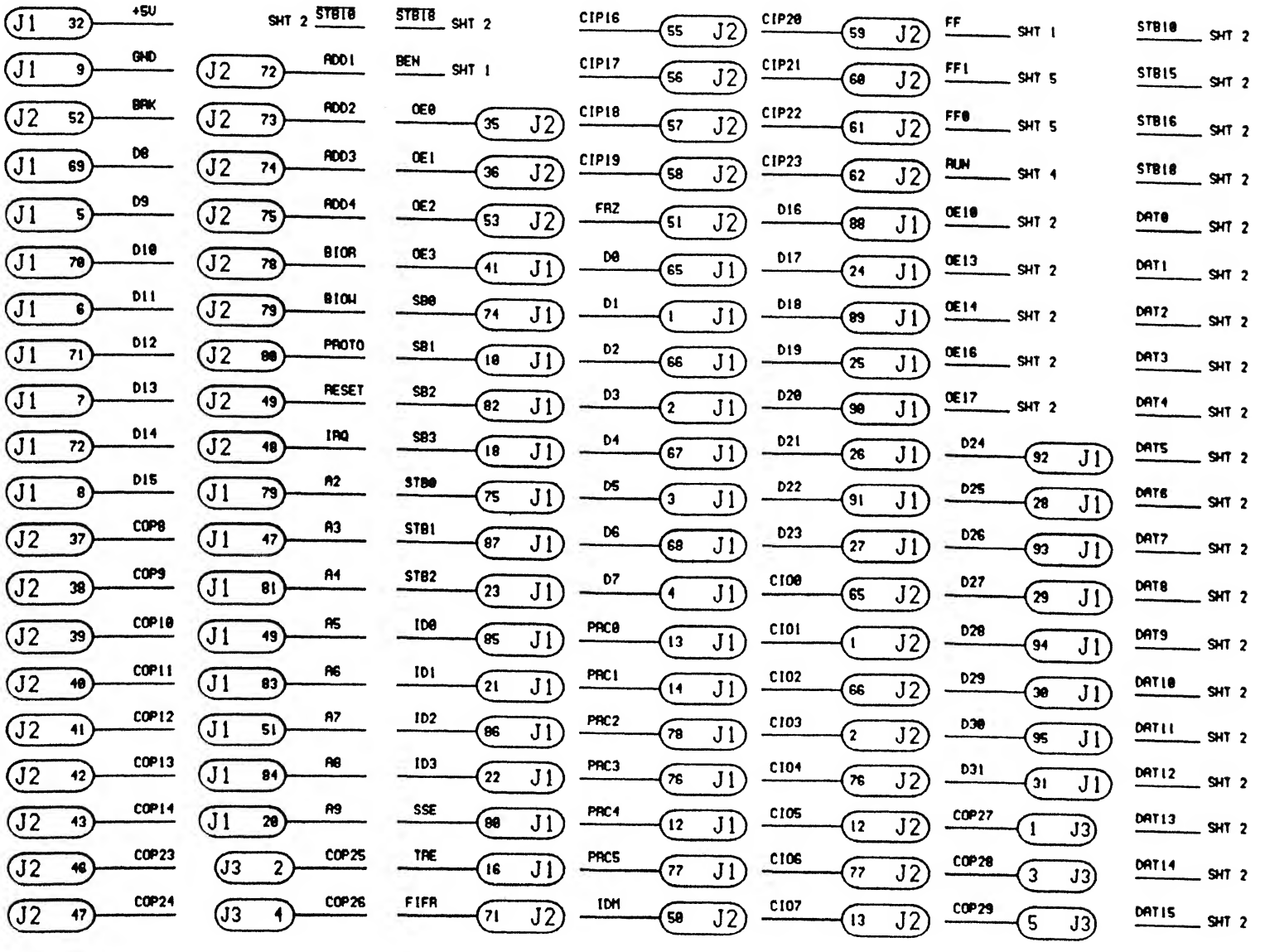


FIGURE D.2 HOST1 CARD: CONNECTIONS SHT 3 of 5

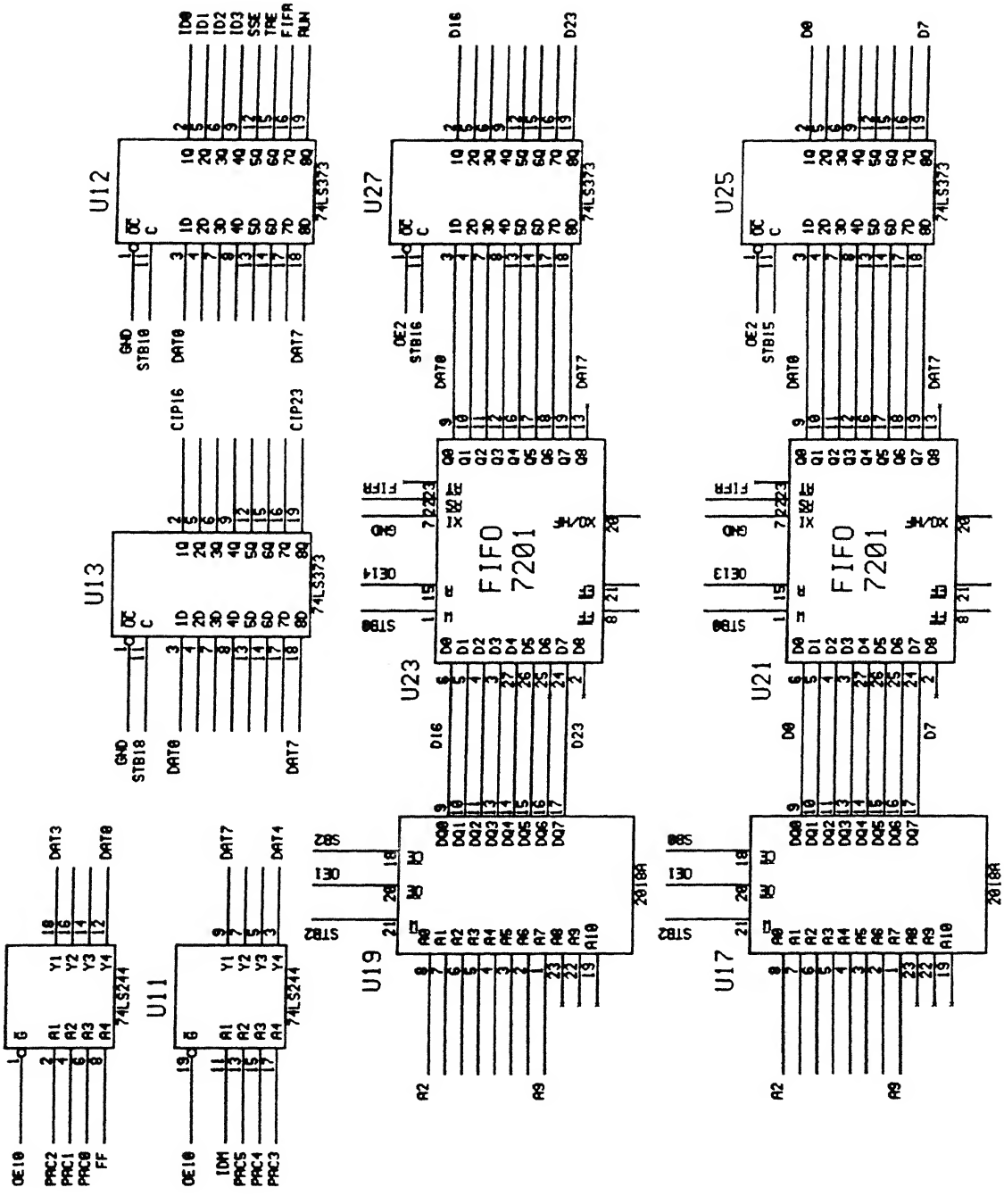


FIGURE D.2 HOST1 CARD SHT 4 of 5

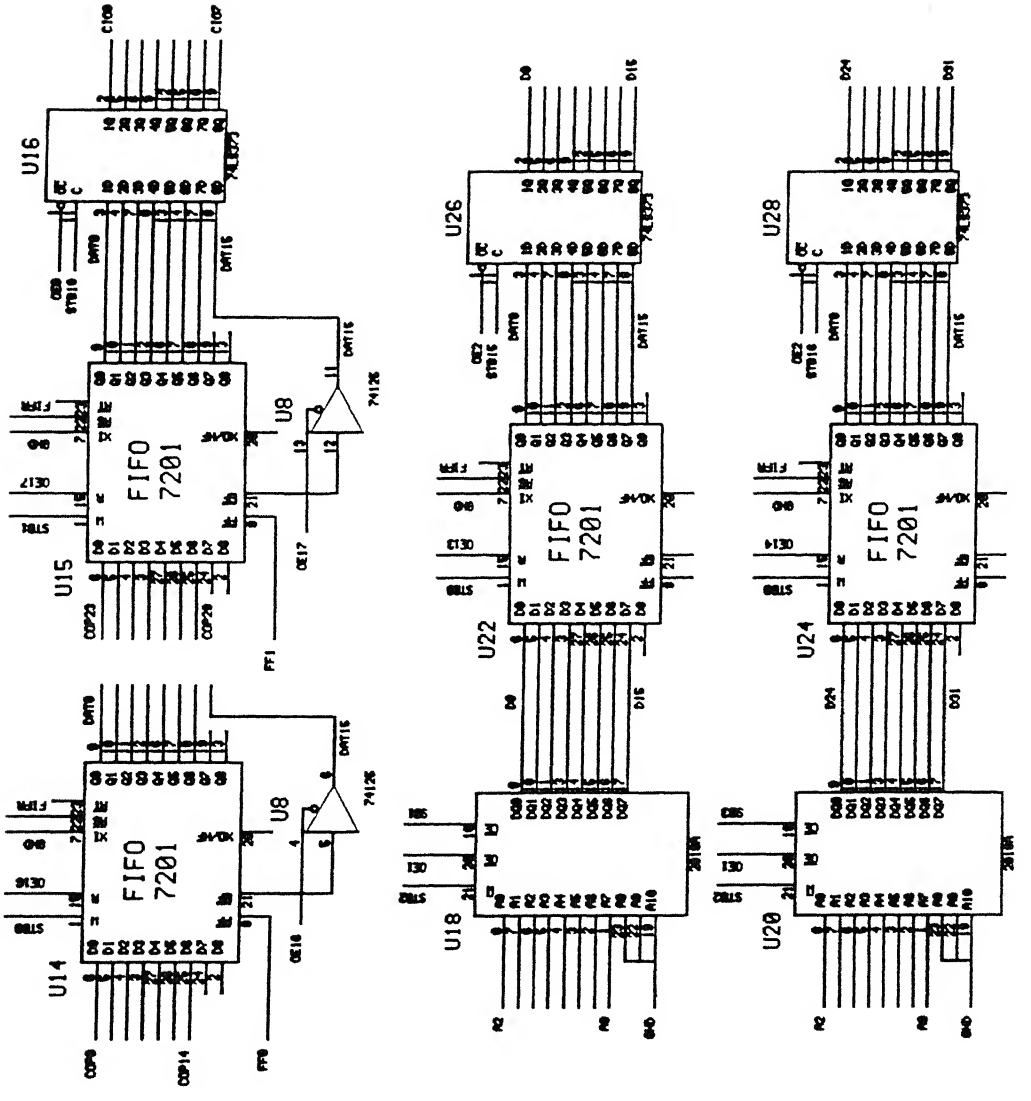


FIGURE D.2 HOST1 CARD SHT 5 of 5

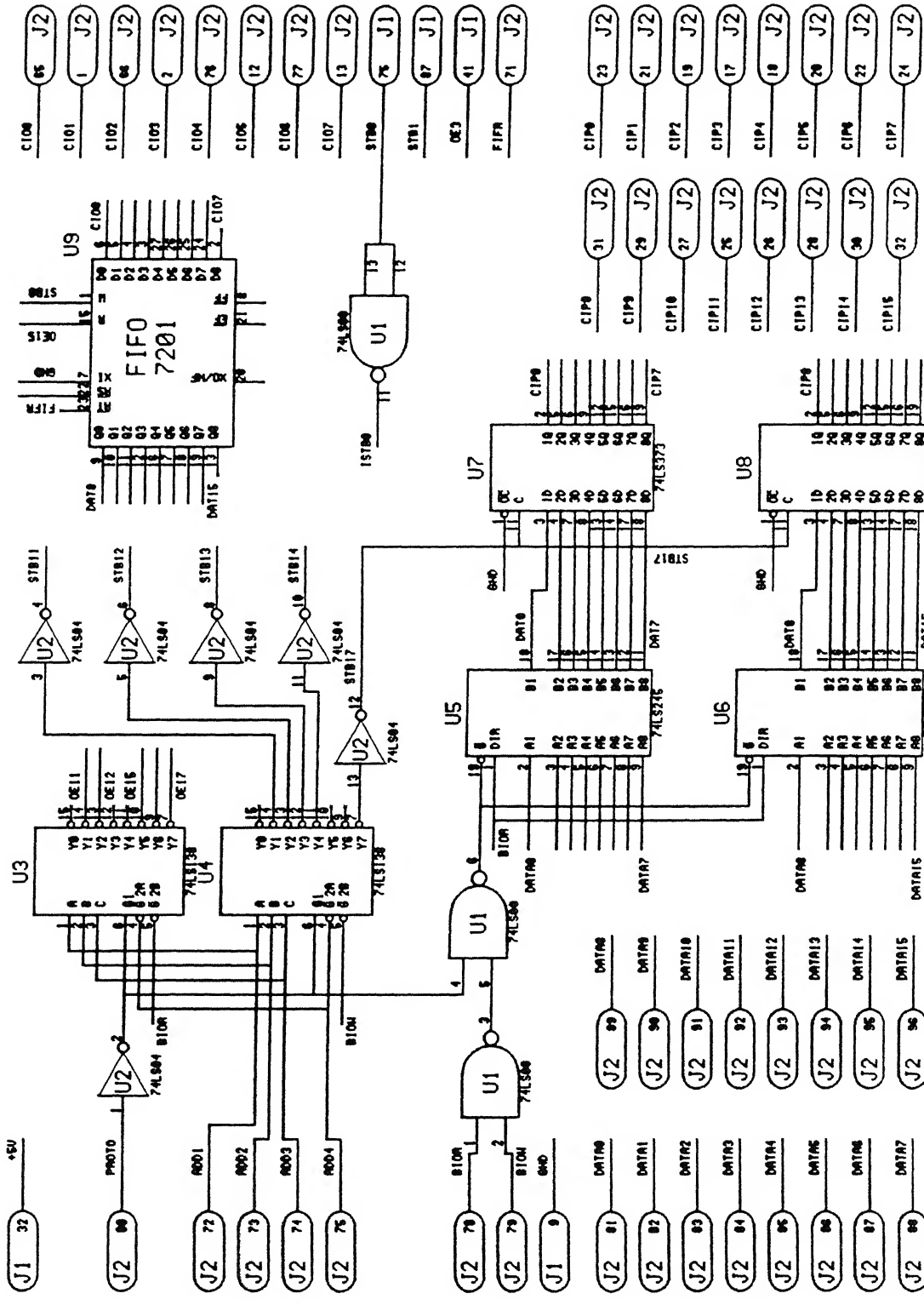


FIGURE D.3

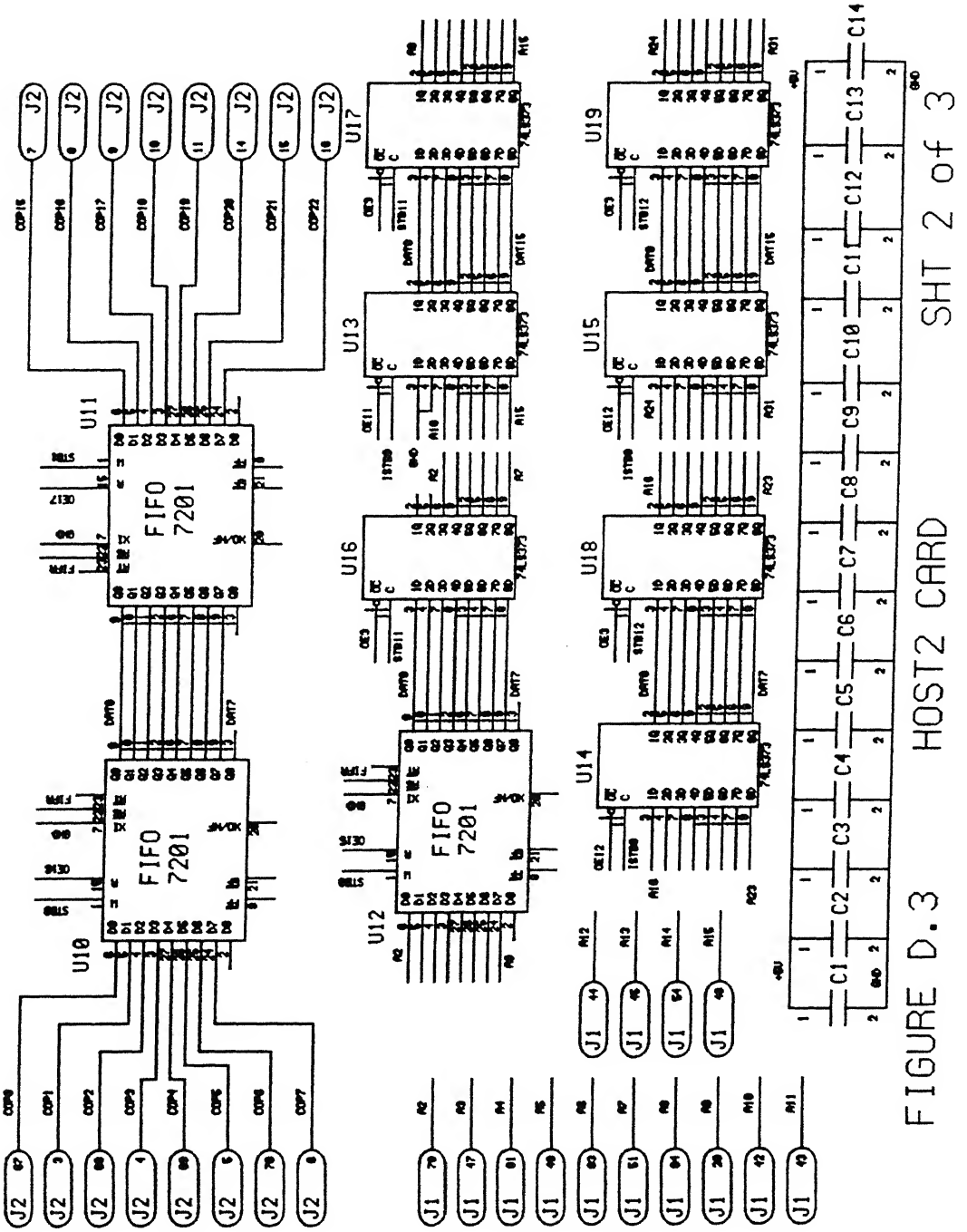


FIGURE D.3 HOST2 CARD SHT 2 of 3

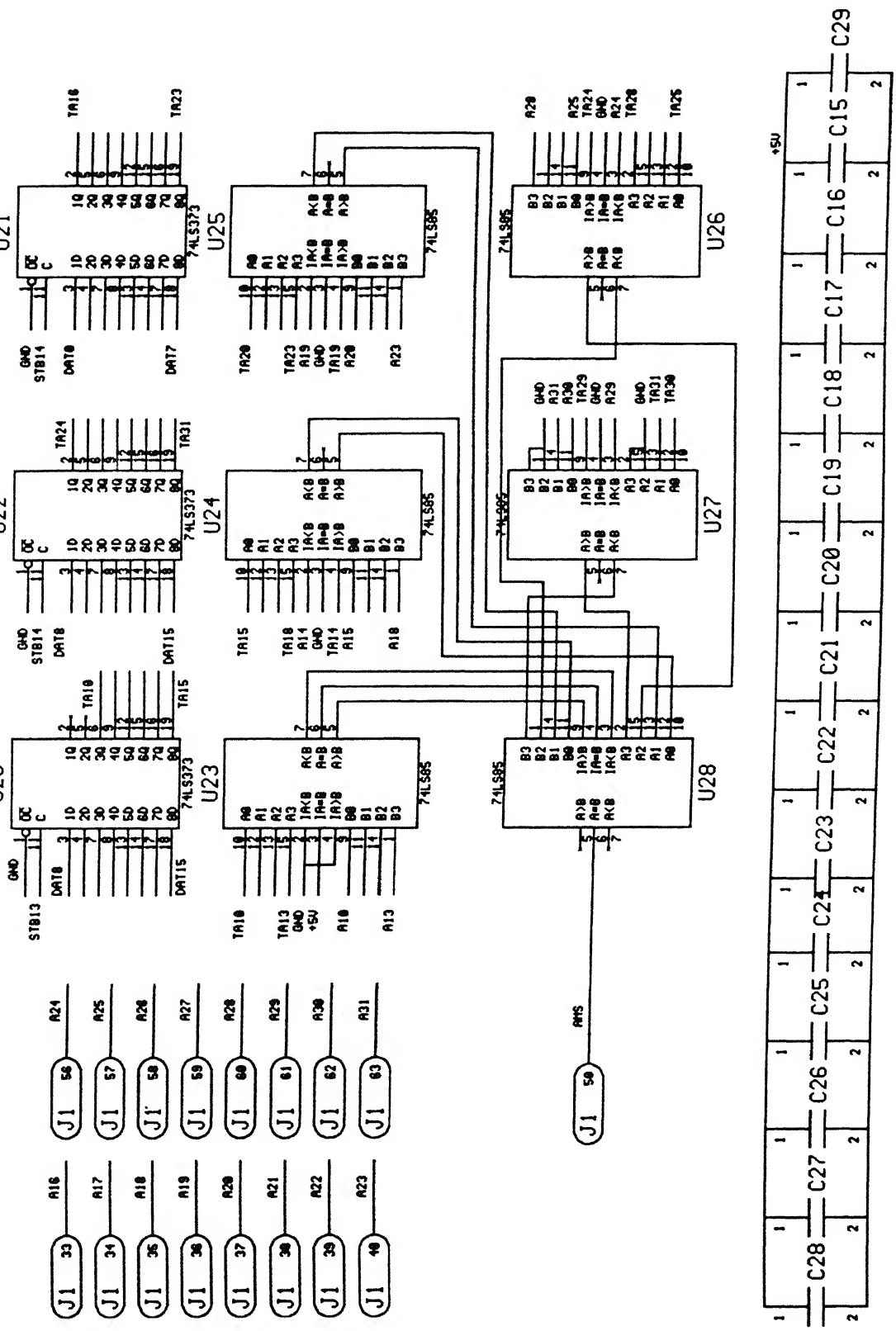


FIGURE D.3 HOST2 CARD SHT 3 of 3

Appendix E

PROGRAMMES AND DISPLAY FORMATS USED IN TEST

E.1 Opening Menu Display Format

OPENING MENU

1. Select new processor
2. Display the allocation map of a processor
3. Go to Run menu
4. Exit

Enter Selection (4)

E.2 Run Menu Display Format

RUN MENU

1. Go to Break-point menu
2. Go to CIP menu
3. Go to Display menu
4. Run
5. Exit to Opening menu

Enter Selection (5)

E.3 Break-point Menu Display Format

BREAK-POINT MENU

1. Page change
2. Single-step execution
3. Continuous trace
4. Continuous run

Enter Selection
Press q to quit

E.4 CIP Menu Display Format

CIP MENU

Press y to activate the corresponding inputs to the uP :
1. Reset out
2. Hold
3. RST 7.5
4. RST 6.5
5. RST 5.5

Press q to quit

E.5 Display Menu Format

DISPLAY MENU

1. Trace only
2. Status only
3. Trace and Status

Enter Selection

Press q to quit